

UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA



ROBUST HANDLING OF OUT-OF-VOCABULARY WORDS IN DEEP LANGUAGE PROCESSING

João Ricardo Martins Ferreira da Silva

DOUTORAMENTO EM INFORMÁTICA
ESPECIALIDADE CIÊNCIAS DA COMPUTAÇÃO

2014

UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA



ROBUST HANDLING OF OUT-OF-VOCABULARY WORDS IN DEEP LANGUAGE PROCESSING

João Ricardo Martins Ferreira da Silva

Tese orientada pelo Prof. Dr. António Horta Branco,
especialmente elaborada para a obtenção do grau de doutor em Informática
(especialidade Ciências da Computação)

2014

Abstract

Deep grammars handle with precision complex grammatical phenomena and are able to provide a semantic representation of their input sentences in some logic form amenable to computational processing, making such grammars desirable for advanced Natural Language Processing tasks.

The robustness of these grammars still has room to be improved. If any of the words in a sentence is not present in the lexicon of the grammar, i.e. if it is an *out-of-vocabulary* (OOV) word, a full parse of that sentence may not be produced. Given that the occurrence of such words is inevitable, e.g. due to the property of lexical novelty that is intrinsic to natural languages, deep grammars need some mechanism to handle OOV words if they are to be used in applications to analyze unrestricted text.

The aim of this work is thus to investigate ways of improving the handling of OOV words in deep grammars.

The lexicon of a deep grammar is highly thorough, with words being assigned extremely detailed linguistic information. Accurately assigning similarly detailed information to OOV words calls for the development of novel approaches, since current techniques mostly rely on shallow features and on a limited window of context, while there are many cases where the relevant information is to be found in wider linguistic structure and in long-distance relations.

The solution proposed here consists of a classifier, SVM-TK, that is placed between the input to the grammar and the grammar itself. This classifier can take a variety of features and assign to words *deep lexical types* which can then be used by the grammar when faced with OOV words. The classifier is based on support-vector machines which, through the use of

kernels, allows the seamless use of features encoding linguistic structure in the classifier.

This dissertation focuses on the HPSG framework, but the method can be used in any framework where the lexical information can be encoded as a word tag. As a case study, we take LX-Gram, a computational grammar for Portuguese, to improve its robustness with respect to OOV verbs. Given that the subcategorization frame of a word is a substantial part of what is encoded in an HPSG deep lexical type, the classifier takes graph encoding grammatical dependencies as features. At runtime, these dependencies are produced by a probabilistic dependency parser.

The SVM-TK classifier is compared against the state-of-the-art approaches for OOV handling, which consist of using a standard POS-tagger to assign lexical types, in essence doing POS-tagging with a highly granular tagset.

Results show that SVM-TK is able to improve on the state-of-the-art, with the usual data-sparseness bottleneck issues imposing this to happen when the amount of training data is large enough.

Keywords natural language processing, supertagging, deep computational grammars, HPSG, out of vocabulary words, robustness

Resumo

(abstract in Portuguese)

As gramáticas de processamento profundo lidam de forma precisa com fenómenos linguísticos complexos e são capazes de providenciar uma representação semântica das frases que lhes são dadas, o que torna tais gramáticas desejáveis para tarefas avançadas em Processamento de Linguagem Natural.

A robustez destas gramáticas tem ainda espaço para ser melhorada. Se alguma das palavras numa frase não se encontra presente no léxico da gramática (em inglês, uma palavra *out-of-vocabulary*, ou OOV), pode não ser possível produzir uma análise completa dessa frase. Dado que a ocorrência de tais palavras é algo inevitável, e.g. devido à novidade lexical que é intrínseca às línguas naturais, as gramáticas profundas requerem algum mecanismo que lhes permita lidar com palavras OOV de forma a que possam ser usadas para análise de texto em aplicações.

O objectivo deste trabalho é então investigar formas de melhor lidar com palavras OOV numa gramática de processamento profundo.

O léxico de uma gramática profunda é altamente granular, sendo cada palavra associada com informação linguística extremamente detalhada. Atribuir corretamente a palavras OOV informação linguística com o nível de detalhe adequado requer que se desenvolvam técnicas inovadoras, dado que as abordagens atuais baseiam-se, na sua maioria, em características superficiais (*shallow features*) e em janelas de contexto limitadas, apesar de haver muitos casos onde a informação relevante se encontra na estrutura linguística e em relações de longa distância.

A solução proposta neste trabalho consiste num classificador, SVM-TK, que é colocado entre o *input* da gramática e a gramática propriamente dita.

Este classificador aceita uma variedade de *features* e atribui às palavras *tipos lexicais profundos* que podem então ser usado pela gramática sempre que esta se depare com palavras OOV. O classificador baseia-se em máquinas de vetores de suporte (*support-vector machines*). Esta técnica, quando combinada com o uso de *kernels*, permite que o classificador use, de forma transparente, *features* que codificam estrutura linguística.

Esta dissertação foca-se no enquadramento teórico HPSG, embora o método proposto possa ser usado em qualquer enquadramento onde a informação lexical possa ser codificada sob a forma de uma etiqueta atribuída a uma palavra. Como caso de estudo, usamos a LX-Gram, uma gramática computacional para a língua portuguesa, e melhoramos a sua robustez a verbos OOV. Dado que a grelha de subcategorização de uma palavra é uma parte substancial daquilo que se encontra codificado num tipo lexical profundo em HPSG, o classificador usa *features* baseados em dependências gramaticais. No momento de execução, estas dependências são produzidas por um analisador de dependências probabilístico.

O classificador SVM-TK é comparado com o estado-da-arte para a tarefa de resolução de palavras OOV, que consiste em usar um anotador morfossintático (*POS-tagger*) para atribuir tipos lexicais, fazendo, no fundo, anotação com um conjunto de etiquetas altamente detalhado.

Os resultados mostram que o SVM-TK melhora o estado-da-arte, com os já habituais problemas de esparses de dados fazendo com que este efeito seja notado quando a quantidade de dados de treino é suficientemente grande.

Palavras-chave processamento de linguagem natural, *supertagging*, gramáticas computacionais profundas, HPSG, palavras desconhecidas, robustez

Agradecimentos

(acknowledgements in Portuguese)

Na capa desta dissertação surge o meu nome, mas a verdade é que muitas outras pessoas contribuíram, de forma mais ou menos direta, mas sempre importante, para a sua realização.

Agradeço ao Prof. António Branco a sua orientação, assim como o cuidado e rigor que sempre deposita nas revisões que faz. É alguém que me acompanha desde os meus primeiros passos na área de Processamento de Linguagem Natural, e com quem aprendi muito sobre como fazer boa investigação.

Estive rodeado de várias pessoas cuja mera presença foi, por si só, em alturas diferentes, uma fonte de apoio e de ânimo: Carolina, Catarina, Clara, Cláudia, David, Eduardo, Eunice, Francisco Costa, Francisco Martins, Gil, Helena, João Antunes, João Rodrigues, Luís, Marcos, Mariana, Patricia, Paula, Rita de Carvalho, Rita Santos, Rosa, Sara, Sílvia e Tiago. Muito obrigado a todos.

Uma tese açambarca muito da vida do seu autor. Felizmente, tive sempre um porto seguro onde retornar todos os dias, ancorar (já que estamos numa de metáforas marítimas), e descansar. Agradeço aos meus pais e ao meu irmão, pelo apoio incessante.

Finalmente, agradeço à Fundação para a Ciência e Tecnologia. Foi devido ao seu financiamento, quer através da bolsa de doutoramento que me atribuíram (SFRH/BD/41465/2007), quer através do seu apoio aos projetos de investigação onde participei, que pude levar a cabo este trabalho.

Contents

Contents	vii
List of Figures	xi
List of Tables	xiii
1 Introduction	1
1.1 Parsing and robustness	2
1.2 The lexicon and OOV words	3
1.3 The problem space	4
1.4 Deep grammars and HPSG	7
1.5 LX-Gram	8
1.6 Goals of the dissertation	11
1.7 Structure of the dissertation	13
2 Background	15
2.1 Lexical acquisition	16
2.1.1 The Lerner system	16
2.1.2 After Lerner	19
2.1.3 Deep lexical acquisition	20
2.2 Assigning types on-the-fly	22
2.2.1 Supertagging	23
2.2.2 A remark on parse disambiguation	26
2.3 Summary	27

3	Techniques and Tools	29
3.1	Head-Driven Phrase Structure Grammar	29
3.2	SVM and tree kernels	39
3.2.1	An introductory example	43
3.2.2	The tree kernel	43
3.3	Summary	47
4	Datasets	49
4.1	Overview of linguistic representations	50
4.1.1	Syntactic constituency	50
4.1.2	Grammatical dependency	50
4.2	Grammar-supported treebanking	51
4.2.1	Dynamic treebanks	53
4.3	CINTIL DeepBank	54
4.3.1	The underlying CINTIL corpus	55
4.4	Extracting vistas	56
4.4.1	CINTIL PropBank	57
4.4.2	CINTIL TreeBank	67
4.4.3	CINTIL DependencyBank	67
4.5	Assessing dataset quality with parsers	69
4.5.1	Constituency parsers	70
4.5.2	Dependency parsers	73
4.6	Summary	74
5	Deep Lexical Ambiguity Resolution	77
5.1	Evaluation methodology	77
5.2	Preliminary experiments	78
5.2.1	Preliminary sequential taggers	79
5.2.2	Preliminary instance classifier	80
5.2.3	Summary of the preliminary experiments	83
5.3	Sequential supertaggers	83
5.4	Instance classifier	85
5.4.1	SVM-TK	86
5.4.2	Restriction to the top- n types	89
5.4.3	Initial evaluation and comparison	90
5.5	Running over predicted dependencies	92
5.6	Experiments over extended datasets	94
5.7	In-grammar disambiguation	97
5.8	Experiments over another language	99
5.9	Summary	104

6	Parsing with Out-of-Vocabulary Words	107
6.1	LX-Gram with SVM-TK	107
6.1.1	Coverage results	109
6.1.2	Correctness results	110
6.1.3	Discussion	112
6.2	Summary	115
7	Conclusion	117
7.1	Summary	117
7.2	Concluding remarks and future work	121
A	Verbal deep lexical types	125
B	Verb lexicon	131
	References	137

List of Figures

1.1	Adding gender agreement to NPs in a CFG	7
2.1	Some elementary structures for <i>likes</i> in LTAG	23
3.1	Typed feature structure	30
3.2	Type definitions for toy grammar	31
3.3	Type hierarchy for toy grammar	31
3.4	Lexicon for toy grammar	32
3.5	Feature structure for an NP rule	33
3.6	Feature structure for the NP “the dog”	34
3.7	Lexicon with SCFs	35
3.8	Feature structure for a VP rule	35
3.9	Encoding the SCF in the lexical type	36
3.10	Maximum margin hyperplane	40
3.11	Two ways of finding the maximum margin hyperplane	42
3.12	A tree and some of its subtrees	44
3.13	Comparing two binary trees	46
4.1	Constituency tree and dependency graph	51
4.2	Snippet of CINTIL Corpus	55
4.3	Example of a fully-fledged HPSG representation	56
4.4	Derivation tree and exported tree from [<code>incr tsdb()</code>]	58
4.5	Exported tree and annotated sentence	59
4.6	Multi-word named entities	60
4.7	Mapping derivation rules to lexical types	61
4.8	A leaf with its full feature bundle	61

4.9	Coordination	62
4.10	Apposition	62
4.11	Unary chains	63
4.12	Null subjects and null heads	64
4.13	Traces and co-indexation	65
4.14	“Tough” constructions	66
4.15	Complex predicate	67
4.16	Control verbs	67
4.17	Extracting dependencies from PropBank	68
4.18	CoNLL format (abridged)	69
4.19	Dependency graph	69
5.1	The 5-word window of C&C on an 8-token sentence	80
5.2	Dependency graph	87
5.3	Trees used by SVM-TK for “encontrar”	88
5.4	Cumulative verb token coverage of top- n verb lexical types	90
5.5	Cumulative token coverage of top- n verbal lexical types	101
5.6	Top- n types needed for a given coverage (LX-Gram vs. ERG)	102
6.1	Mapping from a tag to a generic type	108
6.2	Breakdown of sentences in the extrinsic experiment	111

List of Tables

2.1	SCFs recognized by LERNER	16
2.2	Summary of related work	28
4.1	Out-of-the-box constituency parser performance for v2	72
4.2	Out-of-the-box constituency parser performance for v3	73
4.3	Out-of-the-box dependency parser performance for v3	74
5.1	Features for the TiMBL instance classifier	81
5.2	Accuracy results from the preliminary experiment	83
5.3	Accuracy of sequential supertaggers	84
5.4	Classifier accuracy over all verb tokens	91
5.5	Classifier accuracy over top- n verb types	92
5.6	SVM-TK accuracy over top- n verb types, predicted features	93
5.7	Cumulative size of datasets	94
5.8	Accuracy of sequential supertaggers on extended datasets	95
5.9	Accuracy comparison between SVMTool and SVM-TK (with predicted features), on extended datasets	96
5.10	Accuracy when assigning from the top- n verbal types	98
5.11	Extra memory needed for top- n verbal types	99
5.12	CINTIL DeepBank and Redwoods	100
5.13	SVMTool accuracy on ERG and LX-Gram	103
5.14	Classifier accuracy, top- n verb types, ERG and LX-Gram	103
6.1	Coverage (using SVM-TK for top-10 verbs)	110

Chapter 1

Introduction

The field of Natural Language Processing (NLP) is concerned with the interaction between humans and computers through the use of natural language, be it in spoken or written form.

Achieving this interaction needs an automatic way of understanding the meaning conveyed by a natural language expression. Note that, here, “understanding” is seen as a continuum. Different applications will have different requirements, and while some application manage to be useful with only very shallow processing, others need to rely on deeper semantic representations to fulfill their purpose.

Even within a given application, moving towards a deeper analysis, i.e. one that produces a semantic representation, may lead to improved results. For instance, this is a view that has grown in acceptance over the past few years in Machine Translation, one of the seminal fields in NLP. The methods that currently achieve the best results mostly try to map from one language into another at the level of strings, and are seen as having hit a performance ceiling. Thus, there is a drive towards methods that, in some way, make use of semantic information.

Parsing is one of the fundamental tasks in NLP, and a critical step in many applications. Much like the applications it supports, parsing also lies on a continuum ranging from shallower parsing approaches to deep computational grammars. Accordingly, as applications grow more complex and require deeper processing, parsing must follow suit.

This Chapter provides a short introduction to the topic of parsing and motivates the need for a robust handling of out-of-vocabulary words. The notion of subcategorization frame will then be presented as an introduction to the more encompassing notion of deep lexical type. This is followed by a description of LX-Gram, the particular computational grammar that is used in this work. The Chapter ends with a description of the research goals of this dissertation.

1.1 Parsing and robustness

Parsing is the task of checking whether a sentence is syntactically correct. More formally, parsing consists in answering the so-called “membership problem” for a string of symbols, that is whether such a string is a member of a given set of strings, which constitute a language.

The language, thus taken as a set of valid sentences, is defined through a grammar, a finite set of production rules for strings. Note that, while a grammar is finite, the set of strings it characterizes may be infinite, in which case the membership problem cannot be reduced to a look-up in a list of well-formed sentences.

Parsing proceeds by assigning a syntactic analysis to the sentence being checked. If a full parse is found, the sentence is syntactically correct and thus a valid member of the language.

Many applications can benefit from parsers that have some degree of *robustness* to not fully well-formed input. That is, parsers that, when faced with input with some level of ungrammaticalness, can nonetheless produce useful output. The extent of this robustness, and what counts as being an useful output, depend on the purpose of the application.

For instance, parsers for programming languages are strict and reject any “sentence” (i.e. program) that is not part of the language. However, those parsers usually include some sort of recovery mechanism that allows them to continue processing even when faced with a syntactic error so that they can provide a more complete analysis of the source code and report on any additional errors that are found, instead of failing and outright quitting parsing upon finding the first syntax error.

Applications for NLP also benefit from being robust. This should come as no surprise since robustness is an integral feature of the human capability for language. For instance, written text may contain misspellings or missing words, while speech is riddled with disfluencies, such as false starts, hesitations and repetitions, but these issues do not generally preclude us from understanding that text or utterance. In fact, we are mostly unaware of their presence.

As such, concerning NLP applications, robustness is, more than a matter of convenience, a fundamental property since the input to those applications will often be ungrammatical to a certain degree.

Approaches have been studied to tackle these problems, the more common being (i) shallow parsing and (ii) stochastic methods.

Shallow (or partial) parsing is a family of solutions that hinge on dropping the requirement that in order for a parse to be successful it should cover the whole sentence. In partial parsing, the parser returns whatever chunks, i.e. non-overlapping units, it was able to analyze, forming only a very shallow structure.

Stochastic (or probabilistic) parsing approaches are typically based on an underlying context-free grammar whose rules are associated with probability values. For these approaches, the grammar rules and their probabilities are usually obtained from a training corpus by counting the number of times a rule was applied in the sentences of that corpus. Since it is likely that many grammatical constructions do not occur in the training corpus, these parsing approaches rely on smoothing techniques that spread a small portion of the probability mass to rules not seen during training, though they assume that all rules in the grammar are known (Fouvry, 2003, p. 51). These approaches gain an intrinsic robustness to not fully grammatical input since they are often able to find some rules that can be applied, even if they are ones with low probability, and thus obtain a parse.

These approaches are mostly concerned with robustness regarding the syntactic structure of the input sentences. The focus of the current work is on parser robustness to a different type of issue, that of unknown words in the input sentences.

1.2 The lexicon and OOV words

Most approaches to parsing that build hierarchical phrase structures rely on context-free parsing algorithms, such as CYK (Younger, 1967), Earley chart parsing (Earley, 1970), bottom-up left corner parsing (Kay, 1989) or some variant thereof. There is a great number of parsing methods (see

(Samuelsson and Wirén, 2000) or (Carroll, 2004) for an overview) and all algorithms require a lexical look-up step that, for each word in the input sentence, returns all its possible lexical categories by getting all its lexical entries in the lexicon.

From this it follows that if any of the words in a sentence is not present in the lexicon, i.e. if it is an *out-of-vocabulary* (OOV) word, a full parse of that sentence cannot be produced without further procedures.

An OOV word can result from a simple misspelling, which is a quite frequent occurrence in manually produced texts due to human error. But even assuming that the input to the parser is free from spelling errors, given that novelty is one of the intrinsic characteristics of natural languages, words that are unknown to the parser will eventually occur. Hence, having a parser that is able to handle OOV words is of paramount importance if one wishes to use a grammar to analyze unrestricted texts, in practical applications.

1.3 The problem space

Lexica can vary greatly in terms of the type and richness of the linguistic information they contain. This issue is central to the problem at stake since it is tied to the size of the problem space. As an example of the type of information we may find in a lexicon, we will present the notions of part-of-speech (POS) and subcategorization frame (SCF).¹

A syntactic category, commonly known as part-of-speech, is the result of generalizations in terms of syntactic behavior. A given POS category groups expressions that occur with the same syntactic distribution. That is, an expression with a given POS category can be replaced by any other expression bearing that same category because such replacement preserves the grammaticality of the sentence.²

For instance, in Example (1), the expression *gato* (Eng.: cat) can be replaced by other expressions, such as *cão* (Eng.: dog), *homem* (Eng.: man) or *livro* (Eng.: book) while maintaining a grammatically correct sentence, even if semantically or pragmatically unusual, as in the latter replacement.

- (1) O gato viu o rato
The cat saw the mouse

To accommodate this generalization, these expressions are said to be nouns, or to have or belong to the category Noun.

¹For the sake of simplicity, we present POS and SCF only. Note, however, that the lexica we are concerned with in this dissertation include further information (cf. §3.1).

²Modulo satisfying agreement or subcategorization and selection constraints.

In the same example, *viu* (Eng.: saw) can be replaced by other words, such as *persequiu* (Eng.: chased) or *apanhou* (Eng.: caught), a fact that is generalized by grouping these expressions into the category Verb. However, a more fine-grained observation will reveal that *viu* cannot be replaced by words such as *correu* (Eng.: ran) or *deu* (Eng.: gave), though these are also verbs. This difference in syntactic behavior is captured by the notion of subcategorization, or valence, in which verbs are seen as imposing certain requirements and restrictions on the number and type of expressions that co-occur with them in the same sentence. These expressions are then said to be arguments of the verb, which under this capacity is said to behave as a predicator.

For instance, in Example (1), *viu* can be replaced by *apanhou* because both are transitive verbs, i.e. both require two arguments (of the type noun phrase, in this case). Since the verb *correu* is intransitive (requires one argument) it cannot replace *viu*.

It is important to note that other categories, such as nouns and adjectives, also have SCFs (see, for instance, the work of Preiss *et al.* (2007) on acquiring SCFs for these categories). Nevertheless, the notion of subcategorization is usually introduced with respect to verbs since the words in this category tend to display the richest variety of SCFs.

Information contained in the SCF of words is important in imposing restrictions on what is a well-formed sentence thus preventing the grammar from describing ungrammatical strings. Also, in many cases, a lexicon with SCF information is supplemented with information on the frequency of occurrence of SCFs. This is extremely useful when ranking the many possible analyses of a sentence according to their likelihood.

The granularity of the restrictions imposed by a SCF can vary. In the examples above, only the category of the argument is specified, like when saying that *viu* requires two arguments of type noun phrase. SCFs can be more detailed and capture restrictions on features such as admissible case values, admissible prepositions, etc.

As the detail of the SCF information increases, the lexicon raises increased challenges. If the lexicon is to be built manually, such added granularity will increase the time and amount of work required to create the entries, and what is more crucial, the likelihood of making errors of omission and errors of commission. If some automatic, machine-learning approach to building the lexicon is to be adopted, the added detail will raise data-sparseness issues and increase the likelihood of classification errors.

In any case, the property of novelty of natural languages will not go away, and the need of robust handling of OOV words remains.

Arguments and adjuncts

A SCF encodes restrictions on the number and type of constituents (the arguments) that are required for a given construction to be grammatical. Additionally, it is possible for other constituents to be related to a predictor without them being required by the subcategorization capacity of that predictor. Such constituents are usually called adjuncts or modifiers, and are often expressions related to time, location or other circumstances that are accessory with respect to the type of event described by a predictor.

For instance, note how in Example (2) the modifier *ontem* (Eng.: yesterday) is an optional addition to the sentence and not part of the SCF of *apanhou* (Eng.: caught), as illustrated by its absence in (2-b). This contrasts with (2-c) where the absence of *o rato* yields an ungrammatical structure,³ thus providing evidence of the status of this expression as an argument of *apanhou*.

- (2) a. O gato apanhou o rato ontem
The cat caught the mouse yesterday
b. O gato apanhou o rato
The cat caught the mouse
c. *O gato apanhou
The cat caught

As it often happens with many other empirically-based distinctions, it is not always a clear-cut case whether a constituent is an argument or an adjunct and there has even been work in trying to automatically make this decision, like (Buchholz, 1998, 2002). This discussion is outside the scope of this work. Here, such decisions are implicit in the grammar and in the corpus being used, having been made by the experts that developed the grammar and annotated the corpus.

Subject, direct object, and other grammatical dependencies

Grammatical dependencies describe how words are related in terms of their grammatical function. In Example (2-a), *gato* is the subject of *apanhou*, *rato* is the direct object, and *ontem* is a modifier. As such, grammatical dependencies are closely related to the SCF of words.

These dependencies are usually represented as a directed graph, where the words are nodes, and the arcs are labeled with the grammatical function between those words (cf. §4.1.2).

³The asterisk is used to mark ungrammatical examples.

$\text{NP} \rightarrow \text{Det N}$	$\text{NP} \rightarrow \text{Det}_m \text{N}_m$
$\text{Det} \rightarrow \text{o} \text{a}$	$\text{NP} \rightarrow \text{Det}_f \text{N}_f$
$\text{N} \rightarrow \text{gato} \text{gata}$	$\text{Det}_m \rightarrow \text{o}$
	$\text{Det}_f \rightarrow \text{a}$
	$\text{N}_m \rightarrow \text{gato}$
	$\text{N}_f \rightarrow \text{gata}$
(a) without agreement	(b) with agreement

Figure 1.1: Adding gender agreement to NPs in a CFG

1.4 Deep grammars and HPSG

Deep grammars, also referred to as precision grammars, aim at making explicit grammatical information about highly detailed linguistic phenomena and produce complex grammatical representations of their input sentences. For instance, they are able to analyze long-distance syntactic dependencies and the grammatical representation they produce typically includes some sort of logical form that is a representation of the meaning of the input sentence.

These grammars are sought and applied mainly for those tasks that demand a rich analysis and a precise judgment of grammaticality. That is the case, for instance, in linguistic studies, where they are used to implement and test theories; in giving support to build annotated corpora to be used as a gold standard; in providing educational feedback for students learning a language; or in machine translation, among many of the examples of possible applications.

The simpler grammars used in NLP are supported by an underlying system of context-free grammar (CFG) rules. A limitation of CFGs is that they hardly scale as the grammar is enhanced and extended in order to address more complex and diverse linguistic phenomena.

Compare, for instance, the two tiny CFGs shown in Figure 1.1. The CFG that does not enforce gender agreement, in (a), has fewer rules. However, it will also over-generate syntactic analyses, since it accepts NPs where there is no agreement between the determiner and the noun, which are ungrammatical in Portuguese, like *o gata* (Eng.: the-MASC cat-FEM).

In a plain CFG formalism, the only way to introduce such constraints into the grammar is by increasing the number of non terminal symbols and rules to account for all grammatical combinations of features (Kaplan, 2004, §4.2.2). In these ultra-simplistic examples, adding a feature for gender with

two possible values, masculine or feminine, as in (b), doubled the number of NP rules. As more features are added, the amount of grammar rules quickly becomes unwieldy.

Intuitively, the restrictions imposed by the features that were added are orthogonal to the syntactic constituency structure and should be factored out. That is, it should be possible to have a rule simply stating “a NP is formed by a determiner and a noun (that agree with each other in gender and number).”

Over the years, a number of powerful grammar formalisms have been developed to address the limitations of CFG. For instance, Lexical Functional Grammar (LFG, (Bresnan, 1982)), Generalized Phrase Structure Grammar (GPSG, (Gazdar *et al.*, 1985)), Tree-Adjoining Grammar (TAG, (Joshi and Schabes, 1996)), Combinatory Categorical Grammar (CCG, (Steedman, 2000)), and Head-Driven Phrase Structure Grammar (HPSG, (Pollard and Sag, 1994; Sag and Wasow, 1999)) are grammatical frameworks resorting to different such description formalisms.

The present work uses an HPSG as the underlying linguistic framework. However, it is worth noting that the relevance of the results obtained in the present study is not restricted to this particular framework.

The HPSG formalism will be presented later in §3.1. For the purposes of this introduction it suffices pointing out that each entry in the lexicon of an HPSG grammar is associated with a *deep lexical type* that encodes, among other information, the SCF of the corresponding word and fully specifies its grammatical behavior.

1.5 LX-Gram

LX-Gram (Branco and Costa, 2010) is an HPSG deep grammar for Portuguese that is under development at the University of Lisbon, Faculty of Sciences by NLX—Natural Language and Speech Group of the Department of Informatics.⁴

LX-Gram is the flagship resource produced at NLX, and is well-suited for the goals of the present study due to a number of factors.

The grammar is under continuous development, and in its current state it already supports a wide range of linguistic phenomena.

Its lexicon, with over 25,000 entries, is developed under a design principle of lexicographic exhaustiveness where, for each word in the lexicon, there are as many entries as there are possible distinct syntactic readings (and thus, as many deep lexical types) for that word. In its last stable version, it

⁴LX-Gram is freely available at <http://nlx.di.fc.ul.pt/lxgram/>.

contains over 60 morphological rules, 100 schemas (syntax rules), and 850 lexical types. Looking at the main open categories, these types breakdown into 205 for common nouns, 93 for adjectives and 175 for verbs.

In addition, LX-Gram is distributed together with a companion corpus, CINTIL DeepBank (Branco *et al.*, 2010), a dataset that is highly reliable inasmuch as it is one of the very few deep treebanks constructed under the methodology of double-blind annotation followed by independent adjudication. The process for building the corpus, as well as the corpus itself, will be described in further detail in §4.2 and §4.3.

Finally, this grammar is specially well-documented, being released with a fully-detailed implementation report (Branco and Costa, 2008) that permits to understand the finer details of the coding of the different linguistic phenomena and that provides a fully-fledged characterization of the various dimensions and performance of the grammar.

LX-Gram is being developed in the scope of the Delph-In consortium, an initiative that brings together developers and grammars for several languages, and provides a variety of open-source tools to help deep grammar development.

One such resource is the LinGO Grammar Matrix (Bender *et al.*, 2002), an open-source kit for the rapid development of grammars that provides a cross-language seed computational grammar upon which the initial version of LX-Gram was built.

Grammar coding is supported by the Linguistic Knowledge Builder (LKB) system (Copestake, 2002), an open-source integrated development environment for the development of constraint grammars that includes a graphical interface, debugger and built-in parser.

LKB includes a rather resource-demanding parser. For application delivery, Delph-In provides the PET parsing system (Callmeier, 2000), which is much lighter, robust, portable and available as an API for integration into NLP applications.

Like many other deep computational grammars, LX-Gram uses Minimal Recursion Semantics (MRS) for the representation of meaning (Copestake *et al.*, 2005). This format of semantic representation is well defined in the sense that it is known how to map between MRS representations and formulas of second-order logic, for which there is a set-theoretic interpretation. MRS is described in more detail in Chapter 3, page 37.

Deep computational grammars are highly complex and, accordingly, suffer from issues that affect every intricate piece of software. In particular, given

1. INTRODUCTION

that there can be subtle interactions between components, small changes to a part of the implementation can have far-reaching impact on various aspects of the grammar, such as its overall coverage, accuracy and efficiency. To better cope with these issues, evaluation, benchmarking and regression testing can be handled by the `[incr tsdb()]` tool (Oepen and Flickinger, 1998).

The latter feature, support for regression testing, is particularly useful when developing a large grammar since it allows running a newer version over a previously annotated gold-standard test suite of sentences and automatically find those analyses that have changed relative to the previous version of the grammar, as a way of detecting unintended side-effects of an alteration to the source code.

This is also the tool that supports the manual parse forest disambiguation process, described in §4.2, that was used to build the companion CINTIL DeepBank corpus.

Many deep computational grammars integrate a stochastic module for parse selection that allows ranking the analyses in the parse forest of a given sentence by their likelihood. Having a ranked list of parses allows, for instance, to perform a beam search that, during parsing, only keeps the top- n best candidates; or, at the end of an analysis, to select the top-ranked parse and return that single result instead of a full parse forest.

For the grammars in the Delph-In family, and LX-Gram is no exception, the disambiguation module relies on a maximum-entropy model that is able to integrate the results of a variety of user-defined feature functions that test for arbitrary structural properties of analyses (Zhang *et al.*, 2007).

LX-Gram resorts to a pre-processing step performed by a pipeline of shallow processing tools that handle sentence segmentation, tokenization, POS tagging, morphological analysis, lemmatization and named entity recognition (Silva, 2007; Nunes, 2007; Martins, 2008; Ferreira *et al.*, 2007).

This pre-processing step allows LX-Gram, in its current state, to already have some degree of robustness since it can use the POS information assigned by the shallow tools to handle OOV words. This is achieved by resorting to *generic types*. Each POS category is associated with a deep lexical type which is then assigned to OOV words bearing that POS tag. As such, in LX-Gram a generic type is better envisioned as being a default type that is triggered for a given POS category.

Naturally, the generic (or default) type is chosen in a way as to maximize the likelihood of getting it right, namely by picking for each POS tag the most frequent deep lexical type under that category. Say, considering all

verbs to be transitive verbs. While this is the best baseline choice (i.e. the most frequent type is most often correct), it will nevertheless not offer or approximate the best solution.

1.6 Goals of the dissertation

Deep processing grammars handle with high precision complex grammatical phenomena and are able to provide a representation of the semantics of their input sentences in some logic form amenable to computational processing, making such grammars desirable for many advanced NLP tasks.

The robustness of such grammars still exhibits a lot of room for improvement (Zhang, 2007), a fact that has prevented their wider adoption in applications. In particular, if not properly addressed, OOV words in the input prevent a successful analysis of the sentences they occur in. Given that the occurrence of such words is inevitable due to the novelty that is intrinsic to natural languages, deep grammars need to have some mechanism to handle OOV words if they are to be used in applications to analyze unrestricted text.

Desiderata

The aim of this work is to investigate ways of handling OOV words in a deep grammar.

Whatever process is used to handle OOV words, it should occur on-the-fly. That is, it should be possible to incorporate the grammar into a real-time application that handles unrestricted text (e.g. documents from the Web) and have that application handle OOV words in a quick and transparent manner, without requiring lengthy preliminary or offline processing of data.

There are several approaches to tackling the problem of OOV words in the input (Chapter 2 will cover this in detail).

LX-Gram, for instance, resorts to a common solution by which the input to the grammar is pre-processed by a stochastic POS tagger. The category that the tagger assigns to the OOV word is then used to trigger a default deep lexical type for that category (e.g. all OOV words tagged as verbs are considered to be transitive verbs). While it is true that the distribution of deep lexical types is skewed, this all-or-nothing approach still leaves many OOV words with the wrong type.

Other approaches rely on underspecification, where a POS tagger is again used to pre-process the input and an OOV word is considered to have every deep lexical type that lies under the overarching POS category assigned to

that word. This leads to a much greater degree of ambiguity, which in turn increases the processing and memory requirements of the parsing process, increases the number of analyses returned by the grammar and can also allow it to accept sentences that are ungrammatical as being valid.

In either approach, having the POS category of the OOV word allows the grammar to proceed with its analysis, but at the cost of increased error rate or lack of efficiency.

To overcome this issue, a classifier is needed to automatically assign deep lexical types to OOV words since a type describes all the necessary grammatical information that allows the grammar to continue parsing efficiently, just like if the word had been present in the lexicon all along. Ideally, this classifier should fully eliminate lexical ambiguity by assigning a single deep lexical type to the OOV word.

Current approaches that assign deep lexical types to words use relatively shallow features, typically based on n -grams or limited windows of context, which are not enough to capture some dependencies, namely unbounded long-distance dependencies, that are relevant when trying to find the lexical type of a word. More complex models, capable of capturing such dependencies, must thus be developed, while coping with the data-sparseness that is inevitable when moving to models with richer features.

The approach that is devised and the classifier that is developed should also strive to be as agnostic as possible regarding the specific details of the implementation of the grammar, since these may change as the grammar evolves and also because, by doing so, the same approach can more easily be applied to other grammars.

Sketch of the solution

With this in mind, we now provide a rough sketch of the solution that is proposed, as to provide a guiding thread for this dissertation

This study will be conducted over LX-Gram as the working grammar, which, as a result, will get improved robustness to OOV words though, as mentioned previously, the methodological relevance of the results is not restricted to this particular grammar or specific to the HPSG framework.

LX-Gram, in its current setup, runs over text that is pre-processed by a POS-tagger. We replace this POS-tagger by a classifier that assigns a fully disambiguated deep lexical type to verbs. These types are then used by the grammar when faced with OOV words. That is, instead of having a pre-processing step assign POS tags that are then used to trigger a default type for OOV words, the pre-processing step itself assigns lexical types. This pre-processing step can be run on-the-fly, setting up a pipeline between the

classifier and the grammar.

To encode richer linguistic information, beyond n -grams, the classifier will use features based on linguistic structure, namely grammatical dependencies graphs, since these closely mirror the SCF information that is a large part of what is encoded in a deep lexical type. This requires representing structure as feature vectors for the classifier, which is achieved through the use of tree kernels. This also requires annotating the input to the classifier with a dependency parser.

Finally, though not a goal, we use, as much as possible, existing tools with proven effectiveness.

Summary of goals

The goals of this dissertation are summarized as follows:

- Study efficient methods to handle OOV words in a deep grammar.
- Devise a classifier that is able to assign deep lexical types on-the-fly so that it can be seamlessly integrated into a working deep grammar. The classifier should assign a single deep lexical type to each occurrence of an OOV word, thus freeing the grammar from having to disambiguate among possible types.
- Make use of structured features to improve the performance of the classifier by allowing its model to encode information on grammatical dependencies that cannot be captured when resorting to shallower features, like n -grams or fixed windows of context.

1.7 Structure of the dissertation

The remainder of this dissertation is structured as follows. Chapter 2 covers related work, with an emphasis on lexical acquisition and supertagging.

Chapter 3 provides an introduction to the tools and techniques that are central to our work, namely the HPSG framework and the tree kernels used in SVM algorithms.

Chapter 4 describes the steps that were required in order to obtain the datasets that were used for training and evaluating the classifiers for deep lexical types.

The classifiers are then described and intrinsically evaluated in Chapter 5, while Chapter 6 reports on an extrinsic evaluation task.

Finally, Chapter 7 concludes with a summary of the main points, and some remarks on the results and on future work.

Chapter 2

Background

A productive way of conceptualizing the different approaches to handling OOV words is in terms of the ambiguity space each of these approaches has to cope with.

At one end of the ambiguity-resolving range, one finds approaches that try to discover all the lexical types a given unknown word may occur with, effectively creating a new lexical entry. However, at run-time, it is still up to the grammar using the newly acquired lexical entry to work out which of those lexical types is the correct one for each particular occurrence of that word.

At the other end of the range are those approaches that assign, typically on-the-fly at run-time, a single lexical type to a particular occurrence of an unknown word. Their rationale is not so much to acquire a new lexical entry and record it in the permanent lexicon, but to allow the grammar to keep parsing despite the occurrence of OOV words.

Approaches can also be classified in terms of whether they work offline, typically extracting SCFs from a collection of data; or on-line/on-the-fly, where one or more SCFs are assigned to tokens as needed.

This Chapter presents some related work, starting with offline approaches that acquire new lexical entries with a full set of SCFs, and moving towards on-the-fly approaches that assign a single type.

Subcat. frame	Example
direct object (DO)	<i>greet</i> [DO them]
clause	<i>know</i> [clause I'll attend]
DO & clause	<i>tell</i> [DO him] [clause he's a fool]
infinitive	<i>hope</i> [inf. to attend]
DO & infinitive	<i>want</i> [DO him] [inf. to attend]
DO & indirect object (IO)	<i>tell</i> [DO him] [IO the story]

Table 2.1: SCFs recognized by LERNER

2.1 Lexical acquisition

The construction of a hand-crafted lexicon that includes some kind of SCF information is a resource demanding task. More importantly, by their nature, hand-coded SCF lexica are inevitably incomplete. They often do not cover specialized domains, and are slow to incorporate new words (e.g. the verb *to google* meaning *to search*) and new usages of existing words.

The automatic acquisition of SCFs from text is thus a promising approach for supplementing existing SCF lexica (possibly while they keep being developed) or for helping to create one from scratch.

2.1.1 The Lerner system

The seminal work by Brent (1991, 1993) introduces the LERNER system. This system infers the SCF of verbs from raw (untagged) text through a bootstrapping approach that starts only with the knowledge of functional, closed-class words such as determiners, pronouns, prepositions and conjunctions. It recognizes the 6 SCFs shown in Table 2.1.

Although nearly two decades old, the LERNER system introduced several important techniques, like the use of statistical hypothesis testing for filtering candidate SCFs, that deserve to be covered in some detail.

Since LERNER runs over text that is untagged, it relies on a set of local morphosyntactic cues to find potential verbs. For instance, using the fact that, in English, verbs can occur with or without the *-ing* suffix, it collects words that exhibit this alternation. The resulting list is further filtered by additional heuristics, such as one based on the knowledge that a verb is unlikely to immediately follow a preposition or a determiner.

After the filtering steps, each entry in the remaining list of candidate verbs is then assigned a SCF. To achieve this, the words to the right of the

verb are matched against a set of patterns, or context cues, one for each SCF. For instance, a particular occurrence of a candidate verb is recorded as being a transitive verb—having a direct object and no other arguments—if it is followed by a pronoun (e.g. *me, you, it*) and a coordinating conjunction (e.g. *when, before, while*). In the end, the lexicon entry that is acquired for a particular verb bears all the SCFs that verb was seen occurring given the corpus available.

Hypothesis testing

The verb-finding heuristics and context cues outlined above are prone to be affected by “noisy” input data and are bound to produce mistakes. Accordingly, the assignment of an occurrence of verb V to a particular SCF cannot be taken as definitive proof that V occurs with that SCF. Instead, it should only be seen as a piece of evidence to be taken into account by some sort of statistical filtering technique.

For this, Brent (1993) makes use of the statistical technique of hypothesis testing. A null-hypothesis is formed whereupon it is postulated that there is no relationship between certain phenomena. The available data is then analyzed in order to check if it contradicts the null-hypothesis within a certain confidence level.

The hypothesis testing method works as follows. The occurrence of mismatches in assigning a verb to a SCF can be thought of as a random process where a verb V has a non-zero probability of co-occurring with cues for frame S even when V does not in fact bear that frame.

Following (Brent, 1993), if V occurs with a subcategorization frame S , it is described as a $+S$ verb; otherwise it is described as a $-S$ verb.

Given S , the model treats each verb V as a biased coin flip (thus yielding binomial frequency data). Specifically, a verb V is accepted to be $+S$ by assuming it is $-S$ (the null-hypothesis). If this null-hypothesis was true, the observed patterns of co-occurrence of V with context cues for S would be extremely unlikely. The verb V is thus taken as $+S$ in case the null-hypothesis does not hold.

If a coin has probability p of flipping heads, and it is flipped n times, the probability of it coming up heads exactly m times is given by the well-known binomial distribution:

$$P(m, n, p) = \frac{n!}{m!(n-m)!} \times p^m(1-p)^{n-m} \quad (2.1)$$

From this it follows that the probability of that same coin coming up

2. BACKGROUND

head m or more times (m^+) can be calculated by the following summation:

$$P(m^+, n, p) = \sum_{i=m}^n P(i, n, p) \quad (2.2)$$

To use this, we must be able to estimate the error rate, π_{-s} , that gives the probability of a verb being followed by a cue for frame S even though it does not bear that frame.

For some fixed N , the first N occurrences of all verbs that occur N or more times are tested against a frame S . From this it is possible to determine a series of H_i values, with $1 \leq i \leq N$, where H_i is the number of distinct verbs that occur with cues for S exactly i times out of N .

There will be some j_0 value that marks the boundary between $-S$ and $+S$ verbs. That is, most of the $-S$ verbs will occur j_0 times or fewer with cues for frame S . Given the binomial nature of the experience, the H_i values for $i \leq j_0$ should follow a roughly binomial distribution.

Error rate estimation then consists of testing a series of values, $1 \leq j \leq N$, to find the j for which the H_i values with $i \leq j$ better fit a binomial distribution.

Having the estimate for the error rate, it is simply a matter of substituting it for p in Equation (2.2). If at least m out of n occurrences of V are followed by a cue for S , and if $P(m^+, n, \pi_{-s})$ is small,¹ then it is unlikely that V is a $-S$ verb, leading to the rejection of the null-hypothesis and accepting V as $+S$.

Evaluation is performed over a dataset of 193 distinct verbs that are chosen at random from the Brown Corpus, a balanced corpus of American English with ca. 1 million words. The output of LERNER is compared with the judgments of a single human judge.

Performance is measured in terms of precision and recall. These are calculated from the following metrics: true positives (tp), verbs judged to be $+S$ by both LERNER and the human judge; false positives (fp), verbs judged to be $+S$ by LERNER alone; true negatives (tn), verbs judged to be $-S$ by both; and false negatives (fn), verbs judged $-S$ by LERNER that the human judged as $+S$. The formulas for precision (p) and recall (r) are as follows:

$$p = \frac{tp}{tp + fp} \quad r = \frac{tp}{tp + fn} \quad (2.3)$$

¹A value of 5% is traditionally used. That is, $P(m^+, n, \pi_{-s}) \leq 0.05$. A smaller threshold increases the confidence with which the null-hypothesis is rejected.

Hypothesis testing is performed with a 0.02 threshold and a value of $N = 100$ is used for error rate estimation. LERNER achieves 96% precision and 60% recall.

2.1.2 After Lerner

Several subsequent works on lexical acquisition have built upon the base concepts introduced in LERNER.

Manning (1993) uses a similar approach but improves on the heuristics that are used to find candidate verbs and assign them to SCFs. Candidate verbs are found by running a POS tagger over the raw text. A finite-state parser then runs over the POS tagged text and the SCFs are extracted from its output. The set of SCF types that are recognizable by the system is increased from 6 to 19.

The program was run over a month of New Your Times newswire text, which totaled roughly 4 million words. From this corpus it was able to acquire 4,900 SCFs for 3,104 verbs (an average of 1.6 SCFs per verb). Lower bounds for type precision and recall were determined by acquiring SCFs for a sample of 40 verbs randomly chosen from a dictionary of 2,000 common verbs. The system achieved 90% type precision and 43% type recall.

In (Briscoe and Carroll, 1997), the authors introduce a system that recognizes a total of 160 SCF types, a large increase over the number of SCFs considered by previous systems. An important new feature is that, in addition to assigning SCFs to verbs, their system is able to rank those SCFs according to their relative frequency. Like in the previous systems, hypothesis testing is used to select reliable SCFs.

The system was tested by acquiring SCFs for a set of 14 randomly chosen verbs. The corpus—which totaled 70,000 words—used for the extraction of SCFs was built by choosing all sentences containing an occurrence of one of those 14 verbs—up to a maximum of 1,000 for each verb—from a collection of corpora. Type precision and recall are, respectively, 66% and 36%.

Though hypothesis testing was widely used, it suffered from well-known problems that were even acknowledged in the original paper (Brent, 1993, §4.2). Among other issues, the method is unreliable for low frequency SCFs.

This lead to several works that focus on improving the hypothesis selection process itself. Particularly important is the work by Korhonen (2002), where verbs are semantically grouped according to their WordNet (Miller, 1995) senses to allow for back-off estimates that are semantically motivated.

2. BACKGROUND

Korhonen finds that the use of semantic information, via WordNet senses, improves the ability of the system to assign SCFs. Namely, the system is tested on a set of 75 verbs for which the correct SCF is known. To 30 of those, the system is unable to assign a semantic class, and achieves 78% type precision and 59% type recall. To the remaining 45 verbs the system is able to assign a semantic class, improving performance to 87% type precision and 71% type recall.

As it happens with many other areas of NLP, most of the existing body of work has targeted the English language. Portuguese, in particular, has had very little research work done concerning SCF acquisition.

Marques and Lopes (1998) follow an unsupervised approach that clusters 400 frequent verbs into just two SCF classes: transitive and intransitive. It uses a simple log-linear model and a small window of context with words and their POS categories. An automatically POS-tagged corpus of newswire text with 9.3 million tokens is used for SCF extraction. The classification that was obtained was evaluated against a dictionary, for 89% precision and 97% recall.

Agustini (2006) also follows an unsupervised approach for acquiring SCFs for verbs, nouns and adjectives. Given a word, its SCFs are inferred from text and described extensionally, as the set of words that that word selects for (or is selected by). No quantitative results are presented for the lexical acquisition task.

2.1.3 Deep lexical acquisition

In many works, the acquisition of a lexicon for a deep processing grammar is seen as a task apart from “plain” lexical acquisition given that a deep grammar requires a lexicon with much richer linguist information. In particular, deep lexical acquisition (DLA) tries to acquire lexical information that includes—and often goes beyond—information on SCFs. This Section covers previous work on lexical acquisition that specifically addresses deep grammars.

Dedicated classifier

Baldwin (2005) tackles DLA for the English Resource Grammar (ERG), an HPSG for English (Flickinger, 2000). He uses an approach that bootstraps from a seed lexicon. The rationale being that, by resorting to a measure of word and context similarity, an unknown word is assigned the lexical types of the known word—from the seed lexicon—it is most similar to.

The inventory of lexical types to be assigned was formed by identifying all open-class types with at least 10 lexical entries in the ERG lexicon which, in the version of the ERG used by Baldwin at the time, amounted to 110 lexical types (viz. 28 nouns, 39 verbs, 17 adjectives and 26 adverbs).²

Being a lexicon meant for a deep grammar, these types embody richer information than simple POS tags. For instance, in ERG the type `n_intr_le` indicates an intransitive countable noun.

Lexical types are assigned to words via a suite of 110 binary classifiers, one for each type, running in parallel. Each word receives the lexical types corresponding to each classifier that gives a positive answer. When every classifier returns a negative answer there is a fall-back to a majority-class classifier that assigns the most likely lexical type for that word class. Note that this assumes that the POS of the unknown word—noun, verb, adjective or adverb—is known.

Baldwin experiments with features of varying complexity, weighing better classifier accuracy against the requirement for a training dataset with a richer and harder to obtain annotation. For instance, one of the simpler models requires only a POS tagged corpus for training and uses features such as word and POS windows with a width of 9 tokens; while the most advanced model requires a training corpus tagged with dependency relations and includes the head-word and modifiers as features.

Evaluation is performed over three corpora (viz. Brown corpus, Wall Street Journal and the the British National Corpus) as to assess the impact of corpus size. Using 10-fold cross evaluation, the acquired lexica are compared against the 5,675 open-class lexical items in the ERG lexicon. From all the methods that were tested, the one that used features from a shallow syntactic chunker achieved the best result, with 64% type f-score (the harmonic mean of type precision and type recall).

Leaving it to the grammar

Deep grammars, as a consequence of their precise description of grammatical phenomena and due to the ambiguity that is inherent to natural languages, are capable of producing, for a single sentence, many valid analyses (the parse forest). Though all the analyses in the parse forest are correct according to the grammar, some are more plausible than others. Thus, recent deep grammars include a disambiguation component that allows ranking the parses in the parse forest by their likelihood. This component can be

²Naturally, this means that lexical types with less than 10 entries in the seed lexicon cannot be learned or assigned. This is justified by the assumption that most unknown words will correspond to one of the high-frequency lexical types.

based on heuristic rules that encode a preference or, more commonly, on a statistical model that has been trained over manually disambiguated data.

Van de Cruys (2006) takes advantage of this feature to perform lexical acquisition by allowing the grammar—in that case, the Alpino grammar for Dutch—be the tool in charge of assigning lexical types to unknown words.

The method works roughly as follows: The Alpino parser is applied to a set of sentences that contain the unknown word. The crucial insight of the method is that the parser starts by assigning a universal type—in practice, all possible open category types—to the unknown word, similarly to what was done by Fouvry (2003). Despite the occurrence of this highly underspecified word, with 340 possible tags, the disambiguation model in Alpino will allow the parser to eventually find the best parse and, consequently, the best lexical type to assign to that word. After applying the parser to a large enough number of sentences, a set of lexical types will have been identified as being the correct types for the unknown word.

This method achieved a type f-score of 75% when evaluated over 50,000 sentences containing words that had been purposely removed from the lexicon of Alpino in order to make them unknown to the grammar.

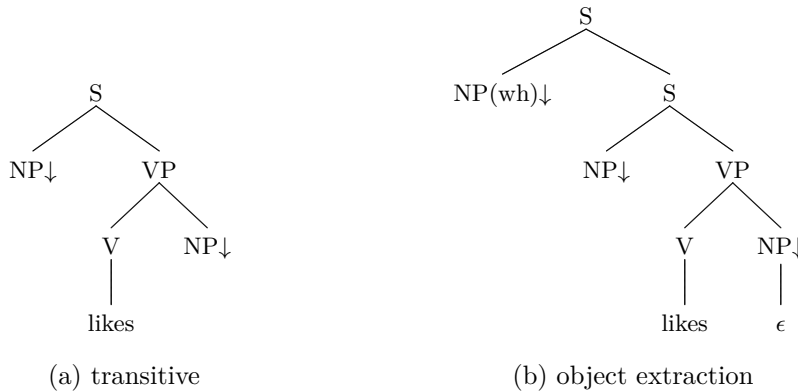
The main problem with this approach is that the universal types, due to their very nature, give much leeway to the grammar, which will likely license many readings that, though grammatically possible, are wrong readings for the unknown word. Another downside is that the additional ambiguity introduced by universal types is likely to lead to greater time and memory requirements for parsing, which would be unsuitable for the on-the-fly OOV handling that is a goal of this work.

2.2 Assigning types on-the-fly

The techniques reviewed so far for acquiring lexicon entries work offline, over a training corpus, looking for unknown words, assigning them a set of SCFs and incorporating the result into the lexicon as new entries.

However, ultimately, such approaches can only acquire the lexicon entries for the unknown words that are present in the training corpus. Thus, any system that is constantly exposed to new text, e.g. when parsing text from the Web, will eventually come across some unknown word that has not yet been acquired.

Moreover, such words must be dealt with on-the-fly, since it is unlikely that the system can afford to wait until it has accumulated enough occurrences of the unknown word to be able to apply any one of the offline lexicon acquisition methods.

Figure 2.1: Some elementary structures for *likes* in LTAG

2.2.1 Supertagging

POS tagging is a task that relies only on local information (e.g. the word itself and a small window of context around it) to achieve a limited form of syntactic disambiguation (Manning and Schütze, 1999, Ch. 10). As such, POS tags are commonly assigned prior to parsing as a way of reducing parsing ambiguity by restricting words to a certain syntactic category. Less ambiguity leads to a greatly reduced search space and, as a consequence, much faster parsing.

Supertagging, first introduced by Bangalore and Joshi (1994), can be seen as a natural extension of this idea to a richer tagset, in particular to one that includes information on subcategorization frames, suited to the level of linguistic detail required by a deep grammar.

In (Bangalore and Joshi, 1994) supertagging was associated with the Lexicalized Tree Adjoining Grammar (LTAG) formalism. As the name indicates, this is a lexicalized grammar, like HPSG, but in LTAG each lexical item is associated with one or more trees, the elementary structures. These structures localize information on dependencies, even long-range ones, by requiring that all and only the dependents be present in the structure. The LTAG formalism then uses two tree rewriting operations—substitution and adjoining—to combine the various elementary structures into a final tree.³

Figure 2.1 shows some of the elementary structure for the verb *likes*. Note how the structure explicitly contains slots, marked by a down arrow, where to plug-in, via the substitution operator, the elementary structures of the NP arguments of the transitive verb.

The supertagger in (Bangalore and Joshi, 1994) assigns an elementary

³See (Joshi, 2004) for an introduction to the LTAG formalism.

2. BACKGROUND

structure to each word using a simple trigram hidden Markov model. The data for training was obtained by taking the sentences of length under 15 words in the Wall Street Journal together with some other minor corpora, and parsing them with XTAG, a wide-coverage grammar for English based on LTAG. In addition, and due to data-sparseness, POS tags were used in training instead of words.

Evaluation was performed over 100 held-out sentences from the Wall Street Journal. For a tagset of 365 elementary trees, this supertagger achieved 68% accuracy.

In a later experiment, Bangalore and Joshi improve the supertagger by smoothing model parameters and adding additional training data (Bangalore and Joshi, 1999). The larger dataset was obtained by extending the corpus from the previous experiment with Penn Treebank parses that were automatically converted to LTAG. The automatic conversion process relied on several heuristics and, though it is not perfect, the authors found that the problematic issues concerning the conversion process were far outweighed by the gains in accuracy that come from the increase in training data.

The improved supertagger increased accuracy to 92% (Bangalore and Joshi, 1999). The supertagger can also assign the n -best tags, which increases the chances of it assigning the correct supertag at the cost of leaving more unresolved ambiguity. With 3-best tagging, it achieved 97% accuracy.

A maximum entropy supertagger was used by Clark and Curran (2003, 2004, 2007) for a Combinatory Categorical Grammar (CCG). This formalism uses a set of logical combinators to manipulate linguistic constructions. For our purposes here, it matters only that lexical items receive complex tags that describe the constituents they require to create a well-formed construction. For instance, the transitive verb *likes* will be assigned the tag $(S \backslash NP) / NP$, meaning that it is a functor that receives an NP argument to its right and returns $S \backslash NP$ which, in turn, is a functor that accepts an NP argument to its left and returns a sentence, S .

The set of 409 lexical categories to be assigned was selected by taking those categories that occur at least 10 times in sections 02–21 of CCGBank, a version of Penn Treebank automatically annotated by CCG.

Evaluation was performed over section 00 of CCGBank, and achieved 92% accuracy.

As with the LTAG supertagger, assigning more than one tag can greatly increase accuracy. However, instead of setting a fixed n -best number of tags—which might be too low, or too high, depending on the case at hand—the CCG supertagger assigns all tags with a likelihood within a factor β of the best tag. A value for β as small as 0.1, which results in an average of 1.4 tags per word, is enough to boost accuracy up to 97%.

Supertagging in HPSG

There has been some work on using supertagging together with the HPSG framework. As with other works on supertagging, it is mostly concerned with restricting the parser search space in order to increase parsing efficiency, and not specifically with the handling of OOV words.

In HPSG,⁴ each entry in the lexicon of the grammar is associated with a deep lexical type, which roughly corresponds to a LTAG elementary tree or a CCG category (Dridan, 2009, §2.3.3 and §4.1). These deep lexical types are the tags assigned by an HPSG supertagger.

Prins and van Noord (2003) present an HMM-based supertagger for the Alpino Dutch grammar. An interesting feature of their approach is that the supertagger is trained over the output of the parser itself, thus avoiding the need for a hand-annotated dataset.

The supertagger was trained over 2 million sentences of newspaper text parsed by Alpino. A gold standard was created by having Alpino choose the best parse for a set of 600 sentences. The supertagger, when assigning a single tag (from a tagset with 2,392 tags), achieves a token accuracy close to 95%.

It is not clear to what extent these results can be affected by some sort of bias in the disambiguation module of Alpino, given that both the sequence of lexical types in the training dataset and in the gold standard are taken from the best parse produced by Alpino.

Matsuzaki *et al.* (2007) use a supertagger with the Enju grammar for English. The novelty in their work comes from the use of a CFG to filter the tag sequences produced by the supertagger before running the HPSG parser. In this approach, a CFG approximation of the HPSG is created. The key property of this approximation is that the language it recognizes is a superset of the parsable supertag sequences. Hence, if the CFG is unable to parse a sequence, that sequence can be safely discarded, thus further reducing the amount of sequences the HPSG parser has to deal with.

The provided evaluation is mostly concerned with showing the improvement in parsing speed. Nevertheless, the quality of the supertagging process can be inferred from the accuracy of the parse results, which achieved a labeled precision and recall for predicate-argument relations of 90% and 86%, respectively, over 2,300 sentences with up to 100 words in section 23 of the Penn Treebank.

Blunsom (2007, §7) focuses on a supertagging approach to deep lexical acquisition for HPSG. Though an offline approach, it can be trivially extended to run on-the-fly.

⁴The HPSG framework will be presented later in §3.1.

A supertagger with a tagset of 615 lexical types was trained over 10,000 sentences and tested over 1,798 test sentences from the treebank associated with ERG. It achieves an accuracy of 91% per token, though it drops to 41% when evaluating only over unknown words.

Also for the ERG, Dridan (2009, §5) tests two supertaggers, one induced using the TnT POS tagger (Brants, 2000) and the other using the C&C supertagger (Clark and Curran, 2003, 2004, 2007), over different datasets that vary in genre and size. For the sake of brevity, we only reference the results that were obtained by running TnT over a dataset of 814 sentences of tourism data.

Dridan experiments with various tag granularities in order to find a balance between tag expressiveness and tag predictability. For instance, assigning only POS—a tagset with only 13 tags—is the easiest task, with 97% accuracy, while a highly granular supertag formed by the lexical type concatenated with any selectional restriction present in the lexical entry increases the number of possible tags to 803, with accuracy dropping to 91%. There is a noticeable impact in accuracy, which drops to 31%, when we look only at performance over unknown words while assigning the highly granular supertags.

2.2.2 A remark on parse disambiguation

Before moving on to the next Chapter, it is important to note that having a process that handles OOV words by automatically assigning deep lexical types to them cannot be the be-all and end-all of grammatical disambiguation. This is clearly highlighted in a study performed, again over the ERG, by Toutanova *et al.* (2002).

In that study, the authors concluded that lexical information accounts for roughly half of the grammatical ambiguity. They show that, even if deep lexical types are perfectly assigned by an oracle supertagger, the disambiguation module of the ERG only picks the correct parse as its first choice for 55% of the sentences. That is, for the remaining sentences, the correct parse ends up in a rank other than first.

This places an upper-bound on the extent of *grammatical disambiguation* that is possible thanks to lexical information alone. However, the current study is primarily concerned with the accuracy in assigning deep lexical types, not with whether the top-ranked parse that is eventually output by the grammar is the correct one (nonetheless, grammar disambiguation performance is measured and results are presented in §5.7).

2.3 Summary

Most of the initial work related with the handling of OOV words was concerned with off-line lexical acquisition where a process extracts the SCFs of a word from a corpus in order to add new entries to a lexicon. Recent work has tackled supertagging, sometimes on-the-fly, though mostly with the purpose of speeding up the parser by reducing its search space. Many different approaches and tools were covered in this Section. Table 2.2 is an attempt to summarize this disparate information. Precision and recall scores, when applicable, are combined into an f-score.

reference	description
Brent (1993)	hypothesis testing; 5 SCFs; 74% f-score
Manning (1993)	hypothesis testing; 19 SCFs; 58% f-score
Briscoe and Carroll (1997)...	hypothesis testing; 160 SCFs; ranking of SCFs; 47% f-score
Korhonen (2002)	hypothesis testing; semantic grouping of verbs; 78% f-score for verbs with a semantic class
Marques and Lopes (1998)...	clusters verbs into two SCFs; 93% f-score
Agustini (2006)	SCFs as a set of related words; no quantitative results
Baldwin (2005)	110 lexical types; set of dedi- cated pointwise classifiers; 64% f-score
Van de Cruys (2006)	grammar disambiguation mod- ule resolves a universal type; 75% f-score
Bangalore and Joshi (1999) ..	supertagging; 365 tags; 92% accuracy for 1-best tag
Clark and Curran (2004)	supertagging; 409 categories; 92% accuracy
Prins and van Noord (2003) .	supertagger; 2,392 tags; trained over grammar output; 95% accuracy
Matsuzaki <i>et al.</i> (2007)	supertagger; CFG post-filter; 88% f-score (pred-arg rela- tions)
Blunsom (2007)	supertagger; 615 types; 91% accuracy
Dridan (2009)	supertagger; 803 tags; TnT: 91% accuracy

Table 2.2: Summary of related work

Chapter 3

Techniques and Tools

This Chapter provides a short introduction to the tools and techniques that are more central to this work.

The first Section provides a quick primer on the HPSG framework. The second Section introduces support-vector machines and discusses how tree kernels can be applied to allow a classifier to use structured features.

3.1 Head-Driven Phrase Structure Grammar

This Section provides a short introduction to the main concepts in the HPSG framework being used. For a detailed account of the underlying theory, see (Pollard and Sag, 1994) and (Sag and Wasow, 1999), among others.

HPSG resorts to a constraint-based description formalism that relies on typed feature structures whose properties make it very amenable to computational implementation. A typed feature structure is a directed acyclic graph (DAG) where each node is labeled with a type and arcs are labeled with features. An equivalent representation is the attribute-value matrix (AVM) notation, which will be used throughout this work, since it is more readable than a DAG, specially for large feature structures. Figure 3.1 shows an example of both representations for the same structure.

An AVM represents a set of attributes, each associated with a value. This value can itself be an AVM, allowing for unlimited nesting of feature structures.

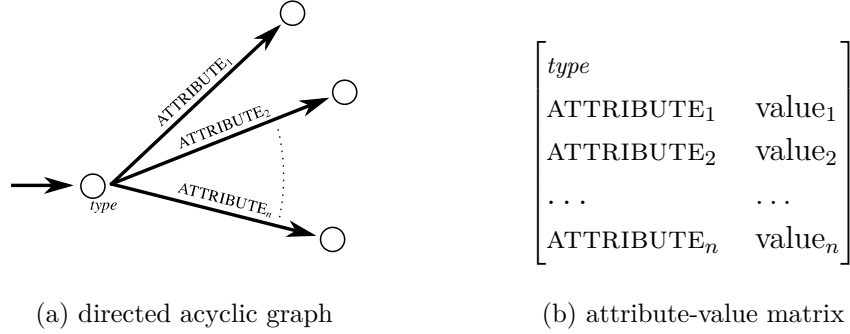


Figure 3.1: Typed feature structure

Each feature structure is assigned a type,¹ which are organized in an inheritance hierarchy, with a unique top node, where a given type inherits the properties of all the types that are more general than (i.e. subsume) it. Roughly speaking, these types state what attributes are present in a feature structure, as well as the allowable types for the values of those attributes.

To better explain how the unification mechanism and the type hierarchy work, the following examples will refer to the toy grammar that is presented in (Copestake, 2002, p. 50).

This grammar defines a *syn-struct* type, which simply states that any feature structure of that type must have a CATEG attribute, with values of type *cat*, and a NUMAGR attribute, with values of type *agr*. These three types, *syn-struct*, *cat* and *agr*, all inherit from the unique top node, **top**.

The type *word* is defined as a *syn-struct* that has an additional attribute ORTH of type *string*. More specific types of *word* are also defined, such as *sg-word* and *pl-word*, for singular and plural words, respectively. These types inherit from *word* and enforce a more specific type for the value of the NUMAGR attribute.

Finally, a *phrase* type is defined as being a *syn-struct* with an ARGS attribute with a *list* value.²

Figure 3.2 summarizes these definitions while Figure 3.3 shows the resulting type hierarchy tree.

Note that some types, namely the ones shown in the right-hand side of Figure 3.2 (e.g. *det* or *sg*), do not place any constraints on the content of

¹More recent HPSG terminology uses the term “sort” instead of “type”, but here I will keep using the latter.

²Lists are also represented as feature structures through the use of a recursive definition, where a list is a first element followed by a possibly empty list. However, to simplify notation, they are often represented using angled brackets.

$\text{syn-struc} \equiv$	$\begin{bmatrix} *top* \\ \text{CATEG} & \text{cat} \\ \text{NUMAGR} & \text{agr} \end{bmatrix}$	$\text{cat} \equiv$	$\begin{bmatrix} *top* \end{bmatrix}$
$\text{word} \equiv$	$\begin{bmatrix} \text{syn-struc} \\ \text{ORTH} & \text{string} \end{bmatrix}$	$\text{det} \equiv$	$\begin{bmatrix} \text{cat} \end{bmatrix}$
$\text{sg-word} \equiv$	$\begin{bmatrix} \text{word} \\ \text{NUMAGR} & \text{sg} \end{bmatrix}$	$\text{n} \equiv$	$\begin{bmatrix} \text{cat} \end{bmatrix}$
$\text{pl-word} \equiv$	$\begin{bmatrix} \text{word} \\ \text{NUMAGR} & \text{pl} \end{bmatrix}$	$\text{np} \equiv$	$\begin{bmatrix} \text{cat} \end{bmatrix}$
$\text{phrase} \equiv$	$\begin{bmatrix} \text{syn-struc} \\ \text{ARGS} & \text{list} \end{bmatrix}$	$\text{agr} \equiv$	$\begin{bmatrix} *top* \end{bmatrix}$
		$\text{sg} \equiv$	$\begin{bmatrix} \text{agr} \end{bmatrix}$
		$\text{pl} \equiv$	$\begin{bmatrix} \text{agr} \end{bmatrix}$

Figure 3.2: Type definitions for toy grammar

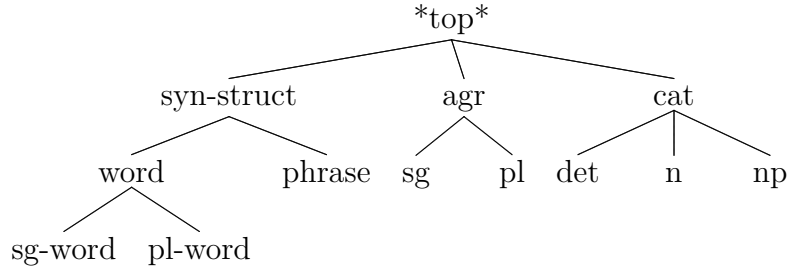


Figure 3.3: Type hierarchy for toy grammar

the feature structure they typify. They are used as placeholders and serve only to indicate the set of acceptable values of a feature.

The grammar lexicon is built using the subtypes of *word*. Given that much of the information about a word can be inferred from its type, only the word-specific information needs to be explicitly specified in the lexical entry. For this toy grammar, this information is just the orthographic form (ORTH) and the grammatical category (CATEG) of the word.

For instance, as shown in Figure 3.4, the lexicon entry “dog” is entered into the lexicon as being of the *sg-word* type and having, as word-specific data, the “dog” orthographic form and the noun category. There are few important remarks regarding this example.

First, it is important not to confuse the name of the lexicon entry with the orthographic form of the word. The fact that, in this case, the name

3. TECHNIQUES AND TOOLS

the \equiv	$\begin{bmatrix} \textit{word} \\ \text{ORTH} & \text{“the”} \\ \text{CATEG} & \text{det} \end{bmatrix}$	fish \equiv	$\begin{bmatrix} \textit{word} \\ \text{ORTH} & \text{“fish”} \\ \text{CATEG} & \text{n} \end{bmatrix}$
this \equiv	$\begin{bmatrix} \textit{sg-word} \\ \text{ORTH} & \text{“this”} \\ \text{CATEG} & \text{det} \end{bmatrix}$	dog \equiv	$\begin{bmatrix} \textit{sg-word} \\ \text{ORTH} & \text{“dog”} \\ \text{CATEG} & \text{n} \end{bmatrix}$
these \equiv	$\begin{bmatrix} \textit{pl-word} \\ \text{ORTH} & \text{“these”} \\ \text{CATEG} & \text{det} \end{bmatrix}$	dogs \equiv	$\begin{bmatrix} \textit{pl-word} \\ \text{ORTH} & \text{“dogs”} \\ \text{CATEG} & \text{n} \end{bmatrix}$

Figure 3.4: Lexicon for toy grammar

of the lexical entry is equal to the value of the ORTH attribute might at first blush make it seem like that feature is useless. Note, however, that the word “dog” is also a transitive verb (cf. to follow, track, pursue). To handle that, a larger grammar, with a more extensive lexicon, would require two separate lexical entries—say, a “dog/1” entry for the noun reading and a “dog/2” entry for the verb reading—with different values for CATEG but with the same value for the ORTH attribute.

Second, note how the lexical entries for “the” and “fish” do not specify whether the word is singular or plural. Instead, both inherit directly from the *word* type and, consequently, from *syn-struct*, giving them a NUMARG attribute with the *agr* value (which subsumes both *sg* and *pl*). In such cases, a value is said to be underspecified.

Lastly, in this toy grammar, one has to explicitly create entries for all the inflectional variants of a word, e.g. “dog” and “dogs”. Not only is this unwieldy, in particular for words with rich inflection (like verbs in Portuguese), it is also redundant since it misses important linguistic generalizations. In a non-toy grammar, this issue is resolved through the use of morphological rules, which allow having in the lexicon only the entry for the lemma of a word and, from it, obtaining all the corresponding inflected forms, while changing the values of the attributes accordingly. For the sake of simplicity, these rules will not be covered here. See (Copestake, 2002, §5.2) for more.

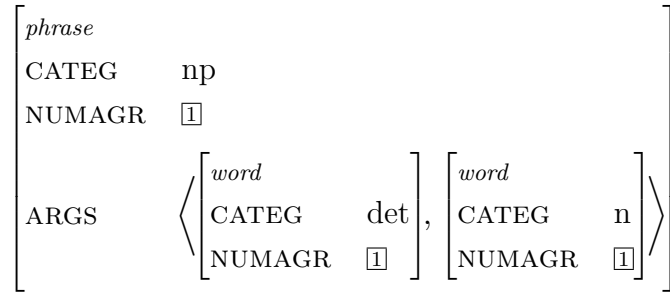


Figure 3.5: Feature structure for an NP rule

So far it might be difficult to find the motivation for such a complex approach and understand why it is useful. The advantage of using an hierarchy of types becomes clearer when one tries to model more complex phenomena and when features structures are combined with each other.

Feature structures are combined through unification. This operation takes two feature structures and merges the information present in each of them into a single feature structure as long as there are no conflicting attribute-value pairs. More formally, the unification of F_1 and F_2 is the most general feature structure that is subsumed by both F_1 and F_2 , if it exists.

HPSG is a highly lexicalized grammar framework, which means that most of the linguistic information is placed in the lexicon. The grammar rules, called schemas, are usually few in number and encode very general linguistic principles.

For instance, retaking the motivating example of the CFG in Figure 1.1, a rule for noun phrases that states that these are formed by a determiner and a noun that must agree with each other can be defined in a straightforward manner, as shown in Figure 3.5.

In more detail, this rule defines a noun phrase as being a subtype of *phrase*, with all that such a type inheritance entails,³ that is formed from two constituents, the first being a determiner and the second a noun. That is, the ARGS list represents what would be the right-hand side in the corresponding CFG rule, $\text{NP} \rightarrow \text{Det N}$.

Features that are marked with matching tags, viz. the $\boxed{1}$ tags in the example, are unified. That is, more than simply being features with matching values, they must be *the same* feature structure. This concept is clearer when looking at the DAG representation of a feature structure, where feature

³That is, a *phrase* is a *syn-struct* that has an ARGS attribute with a *list* value. In turn, being a *syn-struct* also means that it has a CATEG attribute with a *cat* value and a NUMAGR attribute with an *agr* value.

3. TECHNIQUES AND TOOLS

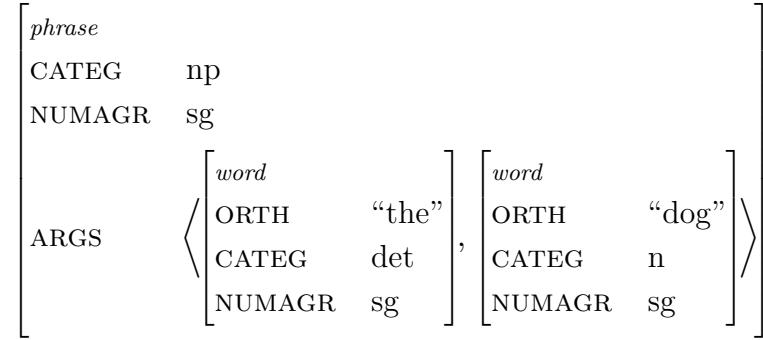


Figure 3.6: Feature structure for the NP “the dog”

co-indexation is explicitly represented through the use of reentrancy, i.e. having several paths that lead to the same node.

In the current example, the practical outcome of co-indexing these features is that it ensures that the determiner and the noun that form a noun phrase must agree, and that the resulting noun phrase will also display that same value for its NUMAGR attribute.

Figure 3.6 shows the feature structure for the noun phrase “the dog”. Note that the NUMAGR attributes of “the” and “dog” can be unified since *agr* subsumes *sg*. As stated before, the result is the most general feature that is subsumed by both, which, in this case, is *sg*.

The manner in which subcategorization constraints are handled is also illustrative of the descriptive power of this framework. To exemplify this, we must move onto richer feature structures for describing words.

Under this richer representation, words have in their feature structure an attribute (here, CAT) that encapsulates syntactic information. This typically contains a HEAD attribute whose value is the syntactic category of the word (noun, verb, preposition, etc.) and, crucially, an attribute (here, SUBCAT) whose value lists the grammatical arguments required by that word, i.e. its subcategorization frame. Two examples are shown in Figure 3.7.⁴

For instance, an intransitive verb like *walks* has a SUBCAT list with a single element whose CAT|HEAD attribute has a value of type *np*, while a transitive verb like *sees* enforces a SUBCAT list with two elements, both with CAT|HEAD attributes whose values are noun phrases.

Having placed the subcategorization information into the lexical entry, the schema for verb phrase rules can then be quite generic. For instance, Figure 3.8 shows a feature structure for a VP rule. Its ARGS list forces

⁴The CAT|HEAD notation is short for a HEAD feature inside a CAT feature.

$$\begin{aligned}
\text{walks} &\equiv \left[\begin{array}{l} \text{word} \\ \text{ORTH} \quad \text{"walks"} \\ \text{CAT} \quad \left[\begin{array}{l} \text{HEAD} \quad \text{v} \\ \text{SUBCAT} \quad \langle [\text{CAT} \mid \text{HEAD} \quad \text{np}] \rangle \end{array} \right] \end{array} \right] \\
\text{sees} &\equiv \left[\begin{array}{l} \text{word} \\ \text{ORTH} \quad \text{"sees"} \\ \text{CAT} \quad \left[\begin{array}{l} \text{HEAD} \quad \text{v} \\ \text{SUBCAT} \quad \langle [\text{CAT} \mid \text{HEAD} \quad \text{np}], [\text{CAT} \mid \text{HEAD} \quad \text{np}] \rangle \end{array} \right] \end{array} \right]
\end{aligned}$$

Figure 3.7: Lexicon with SCFs

$$\left[\begin{array}{l} \text{CAT} \quad [\text{HEAD} \quad \text{vp}] \\ \text{ARGS} \quad \langle [\underline{1}], \left[\text{CAT} \quad \left[\begin{array}{l} \text{HEAD} \quad \text{v} \\ \text{SUBCAT} \quad \langle [\underline{1}], [\underline{2}] \rangle \end{array} \right] \right], [\underline{2}] \rangle \end{array} \right]$$

Figure 3.8: Feature structure for a VP rule

three elements, the second one being a verb with a two-element SUBCAT list (hence, a transitive verb). The co-indexed features ensure that the first and third elements in the ARGS list of the VP must unify with the constraints imposed by the verb.

In the examples just shown, each lexicon entry explicitly states the SCF of the corresponding word. In an actual grammar, this redundancy would be factored out of the lexicon and the corresponding restrictions encoded into the type hierarchy by creating sub-types of *word* that enforce a specific value for their SUBCAT attribute, such as a *verb_np* lexical type which states that the word is an intransitive verb with a single NP argument.

As shown in Figure 3.9, placing this information in the type hierarchy allows grouping under a single lexical type all words with the same SCF, leaving just the word-specific information in the lexical entry, which in these examples amounts only to the ORTH attribute.

Beyond SCFs

For the sake of simplicity, we have been focusing mostly on the lexical information that represents the SCF, but it is important to again note that

$$\text{verb_np} \equiv \left[\begin{array}{c} \text{word} \\ \text{CAT} \left[\begin{array}{cc} \text{HEAD} & \text{v} \\ \text{SUBCAT} & \langle [\text{CAT} \mid \text{HEAD} \quad \text{np}] \rangle \end{array} \right] \end{array} \right]$$

$$\text{walks} \equiv \left[\begin{array}{cc} \text{verb_np} & \\ \text{ORTH} & \text{"walks"} \end{array} \right] \quad \text{sings} \equiv \left[\begin{array}{cc} \text{verb_np} & \\ \text{ORTH} & \text{"sings"} \end{array} \right]$$

Figure 3.9: Encoding the SCF in the lexical type

the deep lexical type encodes a great deal of information not related to the SCF. To illustrate this, we now take a quick look at some examples from the lexicon of LX-Gram.

The lexicographers that build the lexicon of LX-Gram work with a spreadsheet-like database that has one entry per line, with the lexical information given in various columns. The database of verbs, for instance, has 11 columns in addition to the ones that encode SCF information. These columns store information on verb properties like the following:

- Is the verb passivizable? A true/false value that indicates whether the verb can form passive constructions. For instance, the verb *to see* is passivizable while *to arrive* is not.
 - (1) a. Passivizable: *ver* (Eng.: to see)
 - (i) O gato viu o pássaro
the cat saw the bird
 - (ii) The pássaro foi visto pelo gato
the bird was seen by-the cat
 - b. Not passivizable: *chegar* (Eng.: to arrive)
 - (i) O correio chegou
the mail arrived
 - (ii) *O correio foi chegado
the mail was arrived
- Does the verb need the “se” inherent clitic? This has three possible values: obligatory, optional or not allowed.
 - (2) a. Obligatory: *suicidar* (Eng.: to suicide)
 - (i) *Ele suicidou
he suicided

- (ii) Ele suicidou-se
he suicided-SE
“He committed suicide”
- b. Optional: *derreter* (Eng.: to melt)
 - (i) A manteiga derreteu
the butter melted
 - (ii) The butter derreteu-se
the butter melted-SE
- c. Not allowed: *sorrir* (Eng.: to smile)
 - (i) Ele sorriu
he smiled
 - (ii) *Ele sorriu-se
he smiled-SE

- What is the referential opacity of the verb? This property is related to the semantic interpretation of the verb. It can have two values, transparent and opaque. The difference will be explained through the following scenario: Say that John mistakenly believes that Antonio Salieri is the composer of Requiem (the actual composer is Mozart). With a transparent verb, like *to see*, the statement “John saw Mozart” implies “John saw the composer of Requiem”. However, with an opaque verb, like *believe*, the implication is not valid. For instance, “John believes that Mozart arrived” does not imply “John believes that the composer of Requiem arrived” since John mistakenly believes that Salieri is the composer.

These properties, and many more (cf. Appendix B), are specified for each verb entry. The different combinations of the values of such properties are then encoded as a deep lexical type known to LX-Gram. Note that the name of the deep lexical type is simply a string given to a node in the type hierarchy. Nevertheless, adhering to good programming practices means that the name should ideally be meaningful.

Representing semantics

The ultimate result of deep analysis is a semantic representation of the meaning of a sentence. This can be done in several ways, but Minimal Recursion Semantics (MRS) (Copestake *et al.*, 2005) is very popular among grammars in the HPSG framework.

3. TECHNIQUES AND TOOLS

Take, for instance, the classic example of the sentence “Every man loves a woman”. There are two readings for this sentence, which differ in the scope of the quantifiers, and can be represented in first-order logic as follows:

$$\forall x \left(\text{man}(x) \rightarrow \exists y \left(\text{woman}(y) \wedge \text{love}(x, y) \right) \right) \quad (3.1)$$

$$\exists y \left(\text{woman}(y) \wedge \forall x \left(\text{man}(x) \rightarrow \text{love}(x, y) \right) \right) \quad (3.2)$$

In (3.1), the universal quantifier has the wider scope (i.e. “every man loves a possibly different woman”), while in (3.2) it is the existential quantifier that outscopes the universal quantifier (i.e. “there is a woman that is loved by all men”).

First-order logic allows us to express properties of objects. With it we can express set membership, like in $\text{man}(x)$, or relations between objects, like in $\text{love}(x, y)$, but natural languages have many expressions that serve to quantify which cannot be symbolized in terms of a logic restricted to the first-order quantifiers \forall and \exists , the classical example being “most” (Barwise and Cooper, 1981).

The usual way of handling this is to use second-order logic, where relations can have relations as arguments, and generalize quantifiers as relations between two sets, viz. restrictor and scope, with a common bound variable. In this approach, the representations corresponding to the readings shown in (3.1) and (3.2) are, respectively:

$$\text{every} \left(x, \text{man}(x), \text{a} \left(y, \text{woman}(y), \text{love}(x, y) \right) \right) \quad (3.3)$$

$$\text{a} \left(y, \text{woman}(y), \text{every} \left(x, \text{man}(x), \text{love}(x, y) \right) \right) \quad (3.4)$$

MRS uses generalized quantifiers in a flat representation where there is no embedding of predicates. Instead, each predicate is assigned a unique handle and these handles are used as arguments of the generalized quantifiers. The MRS representations corresponding to the readings shown in (3.3) and (3.4) are, respectively:

$$\begin{aligned} h1: \text{man}(x), h2: \text{woman}(y), h3: \text{love}(x, y), \\ h4: \text{every}(x, h1, h5), h5: \text{a}(y, h2, h3) \end{aligned} \quad (3.5)$$

$$\begin{aligned} h1: \text{man}(x), h2: \text{woman}(y), h3: \text{love}(x, y), \\ h4: \text{every}(x, h1, h3), h5: \text{a}(y, h2, h4) \end{aligned} \quad (3.6)$$

Note that the only difference between the two readings, in terms of their MRS representation, is in the handles that are used as the third argument (scope) of the predicates for the generalized quantifiers. As such, a single MRS representation that leaves these handles unbounded can stand for both readings:

$$\begin{aligned} h1: \text{man}(x), h2: \text{woman}(y), h3: \text{love}(x, y), \\ h4: \text{every}(x, h1, h6), h5: \text{a}(y, h2, h7) \end{aligned} \tag{3.7}$$

The MRS in (3.7) leaves scope underspecified since $h6$ and $h7$ do not refer to any predicate. However, it is possible to assign different values to these handles in order to obtain the various possible readings. The first reading is obtained by taking $h6 = h5$ and $h7 = h3$, while taking $h6 = h3$ and $h7 = h4$ gives the second reading.⁵

3.2 SVM and tree kernels

The support-vector machine (SVM) is a well known and widely used supervised discriminative machine-learning algorithm for linear binary classification. It is part of the family of kernel-based methods where a general purpose learning algorithm is coupled with a problem-specific kernel function (Cristianini and Shawe-Taylor, 2000).

This Section provides a short introduction to the main concepts behind these classifiers, with a particular focus on how they can be applied to structured features through the use of tree kernels.

Being a linear binary classifier, an SVM separates instances into two sets using a linear hyperplane. Instances that fall on one “side” of this plane are classified as being positive, while the remaining instances are classified as being negative.

The hyperplane that separates the instances is described by a vector \mathbf{w} , orthogonal to the plane, and a parameter b , the bias, such that \mathbf{x} lies on the hyperplane when $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$, where $\langle \mathbf{w}, \mathbf{x} \rangle$ is the dot product⁶ of \mathbf{w} and \mathbf{x} . As such, the SVM classification function can be formalized as shown in (3.8), the result of which is either 1 or -1 , depending on which “side” of the hyperplane instance \mathbf{x} falls on.

⁵Resolving handles is subject to some constraints: all arguments must be specified and a predicate must fill at most one argument position.

⁶Recall that $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$ for $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$.

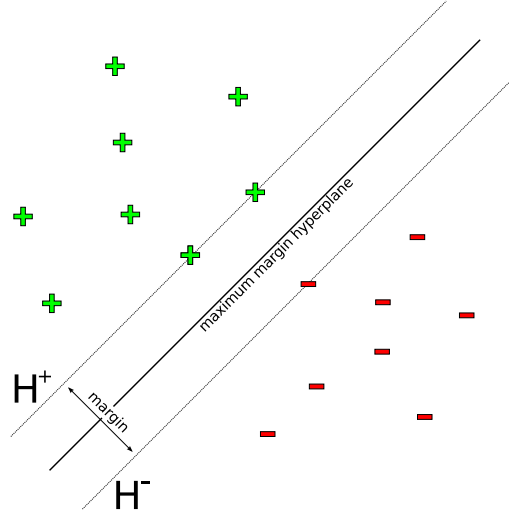


Figure 3.10: Maximum margin hyperplane

$$f_{\mathbf{w},b}(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b) \quad (3.8)$$

There are many possible hyperplanes that separate the positive from the negative instances. SVM finds the hyperplane that gives the maximum margin, the margin being the distance from the hyperplane to the nearest positive and negative instances, as shown in Figure 3.10. The justification for this choice is that, by picking the maximum margin, the classification boundary thus created is more likely to generalize well to future data.

The training data for SVM are pairs of the form (\mathbf{x}_i, y_i) , where \mathbf{x}_i is a vector in \mathbb{R}^n and $y_i \in \{-1, +1\}$ is a label that indicates whether \mathbf{x}_i is a positive or a negative instance. The optimization problem can thus be formulated as follows:

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to:} \quad & \langle \mathbf{w}, \mathbf{x}_i \rangle + b \geq +1 \quad \text{when } y_i = +1 \\ & \langle \mathbf{w}, \mathbf{x}_i \rangle + b \leq -1 \quad \text{when } y_i = -1 \end{aligned} \quad (3.9)$$

That is, we must maximize⁷ the margin between hyperplanes H^+ and H^- , the supporting hyperplanes, subject to the constraints that all positive

⁷The distance between the planes is proportional to the reciprocal of $\|\mathbf{w}\|^2$. Thus, maximizing the margin is equivalent to minimizing $\|\mathbf{w}\|^2$.

instances fall on the positive side of H^+ , and conversely that all negative instances fall on the negative side of H^- . This also means that no instances fall between the supporting planes H^+ and H^- .

A graphical representation of this is given in Figure 3.11(a). Intuitively, the parallel planes H^+ and H^- are pushed apart until they touch the instances. The instances that end up on the supporting hyperplanes are called the support vectors (highlighted with a circle in the figure), and they are particularly important because the solution depends only on them.

The optimization problem (3.9) has a dual form, formulated as follows:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \sum_{i,j} \|p^+ - p^-\|^2 \\ \text{where:} \quad & \\ p^+ = \sum_i \alpha_i \mathbf{x}_i \text{ for } y_i = +1, \quad p^- = \sum_i \alpha_i \mathbf{x}_i \text{ for } y_i = -1 \quad & (3.10) \\ \text{subject to:} \quad & \\ \sum_i \alpha_i = 1 \text{ for } y_i = +1, \quad \sum_i \alpha_i = 1 \text{ for } y_i = -1, \quad \alpha_i \geq 0 \quad & \end{aligned}$$

A graphical representation of this is given in Figure 3.11(b). Intuitively, we take the convex hulls that enclose each of the two sets of instances and find the points p^+ and p^- , one on each hull, that are closest to each other. The separating hyperplane we seek is the one that bisects the line connecting p^+ and p^- .

The optimization problem in (3.10) can be rewritten as follows:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \sum_i \alpha_i \\ \text{subject to:} \quad & \\ \sum_i \alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad & (3.11) \end{aligned}$$

The dual form of the optimization problem has much simpler constraints and, more importantly, the training instances \mathbf{x}_i appear only in the dot product, as the rewritten form in (3.11) shows. This allows applying an SVM to problems that are not linearly separable, as we will see next.

Being limited to a linear separating hyperplane severely restricts the class of problems to which SVM classifiers can be applied since, in many cases, the two sets of points formed by the positive and negative instances cannot be neatly bisected by a hyperplane.

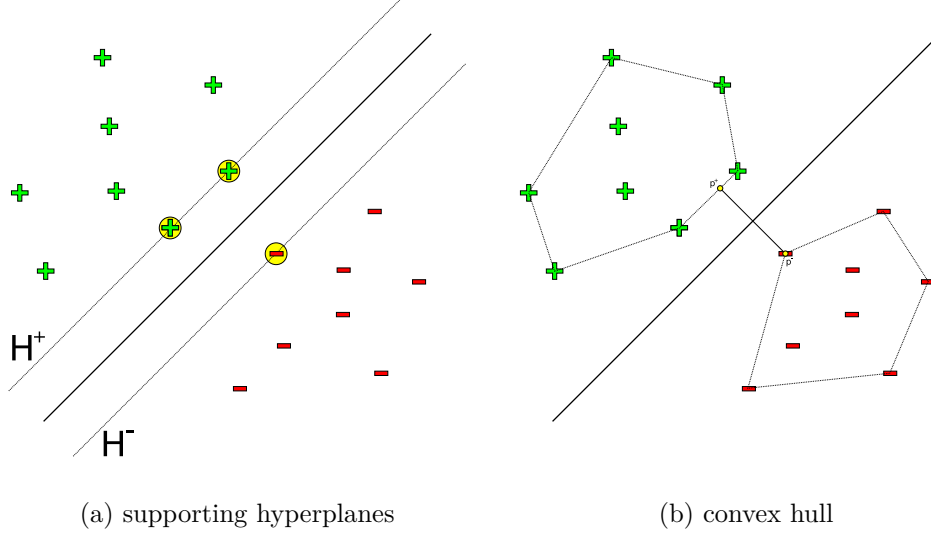


Figure 3.11: Two ways of finding the maximum margin hyperplane

Instead of changing the algorithm, SVM methods tackle this limitation by modifying the data. Data that are not linearly separable are mapped into a new space, usually of much higher dimension, where they can be linearly separated. This is done by extending the original input space with additional dimensions that are obtained through a non-linear transformation. Then, finding a linear separation in this extended space corresponds to finding a non-linear separation in the original input space.

Take, for instance, a problem in \mathbb{R}^2 that requires a circular boundary to separate the positive from the negative instances, a task that clearly cannot be handled by a linear function. However, it is possible to perform a non-linear transformation that maps points in the original two dimensional space into points in a space with five dimensions, $\phi: \mathbb{R}^2 \mapsto \mathbb{R}^5$, in the manner shown in (3.12) (viz. the possible products of the two input features).

$$\phi((x, y)) = (x, y, xy, x^2, y^2) \quad (3.12)$$

A linear hyperplane in this mapped \mathbb{R}^5 feature space corresponds to a quadratic boundary in the input \mathbb{R}^2 space.

Recalling now the dual form shown in (3.11), note that the training data (i.e. the vectors \mathbf{x}_i and \mathbf{x}_j) are involved in the optimization problem only as part of the dot product in the objective function. If the vectors were to be mapped, this dot product is the only place that would be affected and, after mapping, would appear as $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$. This allows the application

of a kernel, which is a function that, for some mappings and under certain conditions, can give the dot product of two mapped points without explicitly knowing the mapping function, as shown in Equation (3.13).

$$\text{ker}(\mathbf{a}_1, \mathbf{a}_2) = \langle \phi(\mathbf{a}_1), \phi(\mathbf{a}_2) \rangle \quad (3.13)$$

Clearly the difficulty lies in defining an adequate mapping and in proving that the conditions that allow applying a kernel hold for the problem at hand. Nevertheless, the kernel property is so surprising at first glance that it is typically referred to as the kernel “trick” or, as Bennett and Campbell (2000, p. 4) put it, “mathematically rigorous magic”.

For the current work, the machine-learning algorithm is to be applied to discrete tree-like structures that encode richer linguistic information, such as syntactic constituency or grammatical dependencies. A suitable kernel for such a task is the tree kernel introduced by Collins and Duffy (2002), which is discussed below.

3.2.1 An introductory example

We begin with a short example from the field of text categorization. Although it does not use tree kernels, it provides some notions that will be used later for the tree kernel.

To use an algorithm like SVM for a text categorization task, we need to first represent documents as (numeric) feature vectors. A simple way of doing this is to consider that a document, D , can be described by a vector that counts the occurrences of the words that form the document (for a concrete example see, for instance, (Joachims, 1998)).

To do this, we start by enumerating all words that occur in the training data. The mapping function, ϕ , represents a document D as a large sparse vector where the number in the i -th position corresponds to the number of occurrences of the i -th word in that document. For instance:

$$\phi(D) = (0, 3, 0, \dots, 0, 5, 0, 2, 0, \dots, 0, 1, 0, \dots, 0, 2, 0)$$

Under this representation, the dot product between the mapping of two documents, $\langle \phi(D_1), \phi(D_2) \rangle$, can be seen as providing a rough measure of their similarity.

3.2.2 The tree kernel

The tree kernel function described here was introduced in (Collins and Duffy, 2002). The underlying idea is that, just like a text can be represented by the

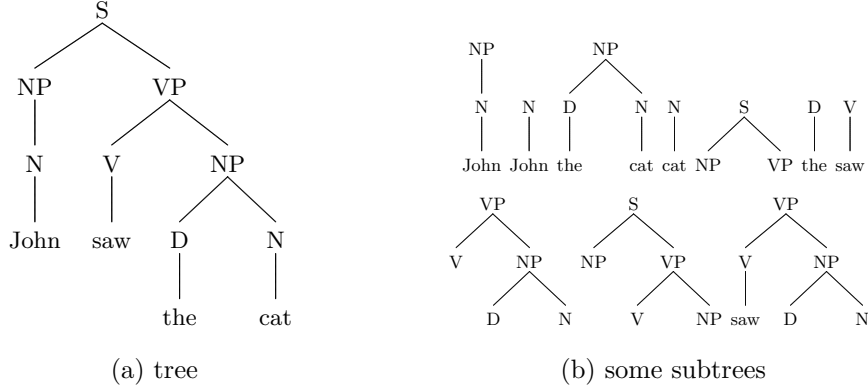


Figure 3.12: A tree and some of its subtrees

words that form it, a tree can be described by all the subtrees that occur in it, as illustrated in Figure 3.12.

This representation starts by enumerating all the subtrees that are found in the training data. A tree, T , is then represented by a huge sparse vector where the number in the i -th position stands for the number of occurrences of the i -th subtree in T .

Similarly to what happens in the text categorization example, under this representation the dot product of two trees can be seen as giving some measure of their similarity. However, explicitly calculating such an operation is prohibitively expensive due to the extremely high number of dimensions involved. Fortunately, the explicit calculation of the dot product can be replaced by a rather simple kernel function that just needs to look at the subtrees that form each tree, as presented next.

Building the tree kernel function

Here we provide a sketch, based on what is presented in (Collins and Duffy, 2002), of how the tree kernel function is obtained.

Start by implicitly enumerating all subtrees that occur in the training data: $1, \dots, n$. Define $h_i(T)$ as being the number of occurrences of the i -th subtree in tree T . From this it follows that the vector representation of trees we wish to obtain is given by $\phi(T) = (h_1(T), h_2(T), \dots, h_n(T))$.

To obtain this mapping, we begin by defining $I_i(n)$, a helper indicator function, which is calculated as shown below.

$$I_i(n) = \begin{cases} 1 & \text{if subtree } i \text{ is rooted at node } n \\ 0 & \text{otherwise} \end{cases} \quad (3.14)$$

From this it follows that $h_i(T_k) = \sum_{n \in N_k} I_i(n)$, where N_k is the set of nodes of tree T_k . That is, $h_i(T_k)$, the number of occurrences of the i -th subtree in tree T_k can be calculated by summing the indicator function of that particular subtree for all the nodes in T_k .

The dot product between the mapping of two trees can be written in terms of h_i and, consequently, in terms of the indicator function defined previously:

$$\langle \phi(T_1), \phi(T_2) \rangle = \sum_i h_i(T_1) h_i(T_2) = \sum_{\substack{n_1 \in N_1 \\ n_2 \in N_2}} \sum_i I_i(n_1) I_i(n_2) \quad (3.15)$$

The outer summation can be seen as going through every pair of nodes from trees T_1 and T_2 , while the inner summation counts the number of common subtrees rooted at those nodes. To simplify the notation, we rewrite the inner summation as $C(n_1, n_2) = \sum_i I_i(n_1) I_i(n_2)$, which leads to the following definition for the kernel function.

$$\text{ker}(T_1, T_2) = \langle \phi(T_1), \phi(T_2) \rangle = \sum_{\substack{n_1 \in N_1 \\ n_2 \in N_2}} C(n_1, n_2) \quad (3.16)$$

That is, the kernel function corresponds to summing the value $C(n_1, n_2)$ for every possible node pairing of the two trees being compared. This function can be defined recursively as shown in (3.17).

$$C(n_1, n_2) = \begin{cases} 0 & \text{if } n_1 \neq n_2 \\ 1 & \text{if } n_1 = n_2, \text{ preterminals} \\ \prod_{j=1}^{nc(n_1)} \left(1 + C(ch(n_1, j), ch(n_2, j)) \right) & \text{if } n_1 = n_2, \text{ phrasal} \end{cases} \quad (3.17)$$

where $nc(n)$ is the number of children of node n and $ch(n, j)$ is the j -th child of n . The notation $n_1 = n_2$ is short for stating that both nodes have the same production rule (e.g. both are $S \rightarrow NP VP$). Conversely, $n_1 \neq n_2$ states that the production rules of nodes n_1 and n_2 are different.

Here lies the step that makes the computation tractable: The sum $\sum_i I_i(n_1) I_i(n_2)$ is calculated over the set of all subtrees, but this has been rewritten as $C(n_1, n_2)$, which only needs to look at the nodes of the two trees being compared.

3. TECHNIQUES AND TOOLS

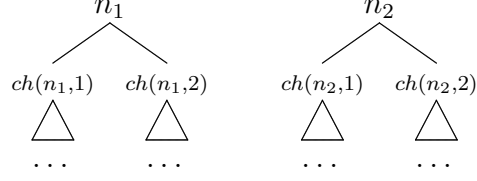


Figure 3.13: Comparing two binary trees

We still need to show that $C(n_1, n_2)$ works as intended and does indeed count the number of common subtrees rooted at n_1 and n_2 . To see that this is so, we will go through each possible case in turn.

The first two cases are trivial: When the productions are different, the number of common subtrees rooted at n_1 and n_2 is 0, since the root productions themselves are different; and when both nodes are preterminals with the same category, there is only 1 subtree in common, namely the preterminal node itself.

The third case, the recursive step, is more complex and thus calls for a more detailed explanation. Its formula is repeated in (3.18).

$$C(n_1, n_2) = \prod_{j=1}^{nc(n_1)} \left(1 + C(ch(n_1, j), ch(n_2, j)) \right) \quad (3.18)$$

The common subtrees between n_1 and n_2 are found by taking the production rule rooted at those nodes and choosing, for each child j , only the child node itself or any of the common subtrees rooted at that child.

Take, for instance, the case shown in Figure 3.13, where two binary branches are being compared. The result of the calculation of the recursive step is shown in (3.19). The number of common subtrees between n_1 and n_2 corresponds to the sum of the following values: 1, from taking only the root production; C_1 , from taking any of the subtrees common to the first children, and no subtrees from the second children; C_2 , from the converse of the previous case; and finally C_1C_2 , from taking any of the subtrees common to the first children combined with any of the subtrees common to the second children.

$$\begin{aligned} C(n_1, n_2) = & \left(1 + \underbrace{C(ch(n_1, 1), ch(n_2, 1))}_{C_1} \right) \times \left(1 + \underbrace{C(ch(n_1, 2), ch(n_2, 2))}_{C_2} \right) = \\ & 1 + C_1 + C_2 + C_1C_2 \quad (3.19) \end{aligned}$$

Kernel methods are quite powerful and flexible. For instance, Collins and Duffy (2002) further refine this tree kernel by adding a decay factor that downweights the effect of large fragments, and it is possible, through minor alterations to how $C(n_1, n_2)$ is defined, to obtain a variant that allows partial productions. For more on this, see (Moschitti, 2006).

3.3 Summary

This Chapter provided a quick and simple overview of the tools that are central to this work and of the theory behind them.

The first Section covered the main features of HPSG that are relevant for the current work. Naturally, this is not the place for an in-depth explanation of this framework, so the examples provided were quite simple and barely scratch the surface of the potential of HPSG. In a wide-coverage, non-toy grammar, like LX-Gram, the rule schemas are very general and are an attempt at encoding actual linguistic universals, valid across human languages. In addition, the lexicon for such grammars is extremely detailed, with entries that encode highly granular SCF constraints, and beyond.

However, for the purposes of this dissertation, the most important fact to retain from this introduction is that the linguistic information that characterizes the grammatical behavior of words is fully specified in the HPSG lexical types themselves. As shown in Figure 3.9, the lexical entry of the word only includes the word-specific information, such as its orthographic form, phonology, etc. That is, knowing the lexical type of a word is enough to fully determine the restrictions that are relevant to the grammar.

The second Section provided an overview to the basic ideas behind support-vector machines. It also introduced tree kernels and a sketch of the proof that shows that the kernel function that was obtained calculates the dot product in a feature space where each tree is represented by all its subtrees.

Chapter 4

Datasets

The present work aims at adding robustness to a deep grammar by increasing its coverage with respect to OOV words, which will be assigned lexical types on-the-fly. The approach that we are pursuing allows us to envisage the type of language resources that are needed.

To be able to use supervised machine learning approaches, one needs linguistically interpreted datasets in order to estimate the relevant stochastic parameters and, at a later step, evaluate the performance of the resulting solution. These datasets will vary greatly in complexity depending on the level of linguistic annotation they contain. Related work (cf. Chapter 2) can clue us into the kind of datasets that are useful, namely a POS-tagged corpus, a constituency treebank and a dependency treebank are all bound to include potentially useful features for the task at hand. In all cases, the datasets must also include information on the lexical types of words, given that this is the information we wish to eventually learn to assign.

This Chapter addresses the organization of the various annotated datasets. It starts with a quick overview of two important linguistic representations, syntactic constituency and grammatical dependencies. This is followed by an introduction to grammar-supported treebanking, which is the method used to build the core dataset, CINTIL DeepBank, from an initial corpus, CINTIL. Then, it covers the extraction of vistas and describes the datasets that result from that process. Finally, the Chapter addresses an assessment of dataset quality carried out via training of data-driven parsers.

4.1 Overview of linguistic representations

This Section provides a short introduction to the linguistic representations that will be used ahead, namely syntactic constituency and grammatical dependencies.

4.1.1 Syntactic constituency

The observation and study of language allows inferring regularities and patterns from how words combine with other words. The formation of syntactic constituents is one such regularity.

In a sequence of words, $w_1w_2w_3$, if the subsequence w_1w_2 has a higher level of aggregation than $w_1w_2w_3$ or w_2w_3 , the sequence w_1w_2 is considered to form a constituent of $w_1w_2w_3$, of which w_1 and w_2 are themselves constituents.

The contrasting levels of aggregation are determined through the application of empirical linguistic tests which rely on grammatical intuitions or judgments on syntactic well-formedness. These empirical tests are based on carefully designed minimal pairs of sequences. To test a putative constituent, these minimal pairs are constructed, for instance, by means of replacing a sequence by another (see (Kim and Sells, 2008, §2.3) for some other common tests). For example, in the sentence “The cat jumped”, the first two words can be replaced by “it”, resulting in “It jumped”, which empirically supports the claim that “The cat” is a constituent of the whole sentence, while “cat jumped” is not.

A syntactic category is a set of constituents with identical syntactic distribution, that is constituents whose replacement by each other constituents of the same category preserves the syntactic well-formedness of larger expressions they are constituents of, modulo other relational constraints such as morphosyntactic agreement, for instance.

A constituent is represented by enclosing the relevant sequence in brackets or, in an alternative but equivalent notation, by forming a one level deep tree whose leaves are w_1 and w_2 and the top node stands for the whole constituent.

4.1.2 Grammatical dependency

Grammatical dependency is an alternative view on how words are combined that resorts to the lexical subcategorization properties of the words.

A given grammatical function (e.g. subject, direct object, etc.) results from an abstraction over complements or modifiers of different predicates. It permits to categorize complements, or modifiers, with similar constraints

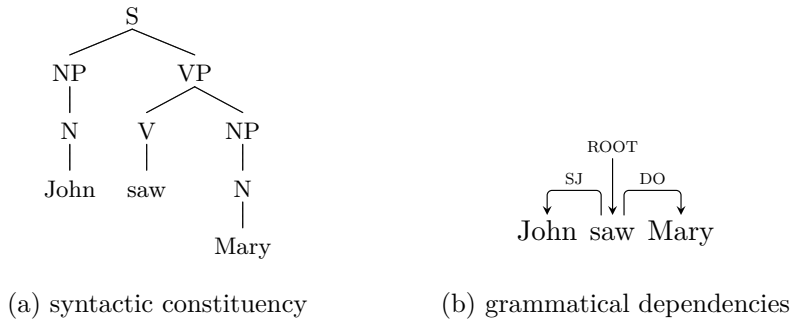


Figure 4.1: Constituency tree and dependency graph

on their realization, such as category, case, agreement, canonical word order, inflection paradigm, etc. As before, these can be determined through the application of empirical tests, like testing for subject-verb agreement (see (Kim and Sells, 2008, §3.2) for some more typical tests).

In an utterance, a word w_b depends on word w_a when the occurrence of w_b in its specific position is made possible by the occurrence of w_a . In such case, it is considered to exist a grammatical dependency relation from the word w_b , the dependent element, to the word w_a , the governor element of the dependency.

Dependency relations can be depicted as graphs whose nodes are words and whose directed arcs establish a connection from governors to their dependent words. An example of a constituency tree and a dependency graph for the same sentence are shown in Figure 4.1.

4.2 Grammar-supported treebanking

Annotated corpora are key resources for NLP. Not only are they important materials for researchers investigating linguistic phenomena, they also allow one to automatically obtain data-driven models of language and evaluate the tools thus produced.

Annotating corpora with human-verified linguistic information is a time-consuming, painstaking and often error-prone task. Early datasets for NLP, like the well-known Penn Treebank (Marcus *et al.*, 1993) or the NEGRA corpus (Skut *et al.*, 1997), were built with the help of automatic annotation tools that were used to provide a preliminary annotation which was then manually corrected. Performing such corrections purely by hand can introduce formatting errors since manual changes may easily be ill-formed (e.g. misspelled categories, forgetting to close a bracket, etc). As

such, the manual correction task itself is often performed with the support of a software tool that ensures that at least the linguistic information is well formed (see, for instance, the works of Marcus *et al.* (1993) or Brants and Plaehn (2000)).

As the linguistic information one wishes to include in a corpus grows in complexity, the direct manual correction of automatically generated annotation becomes increasingly hard to adopt since the human annotator, even with the help of supporting software tools, has to keep track of too much interconnected information.

To address this issue, approaches to corpus annotation have come to rely on the support of an auxiliary deep processing grammar as a way of producing rich annotation that is consistent over its morphological, syntactic and semantic layers. Two examples of such an approach are (Dipper, 2000), using an LFG framework, and (Oepen *et al.*, 2002), under HPSG.

This grammar-supported approach to corpus annotation consists in using a deep processing computational grammar to produce all possible analyses for a given sentence. What is then asked of the human annotator is to consider the output of the grammar and select the correct parse among all those that were returned. In such a setup, the task of the human reviewer can be better envisaged as being one of *manual disambiguation* instead of manual annotation.

Due to the inherent ambiguity of natural language, the parse forest that results from the grammar producing all possible analyses for a sentence may very well include hundreds of trees. Manually examining each individual tree in the parse forest in search for the correct one would prove unfeasible.

To tackle this issue, grammar-supported approaches typically rely on *discriminants*, as proposed by Carter (1997) for the TreeBanker system.

A discriminant is a binary property that can be used to distinguish between competing analyses. That is, a property that holds for some of the analyses but not for others. These discriminants can be automatically associated to the parse forest, yielding a set of binary decisions that allow to unambiguously specify a single analysis from the parse forest.

For instance, PP-attachment is a common source of structural ambiguity, where a PP (prepositional phrase) constituent may validly attach to more than one node in the parse tree. A discriminant would state whether the PP attaches to a given constituent. Choosing that discriminant as being valid automatically prunes from the parse forest all parses where the attachment of that PP is different, while marking the discriminant as invalid discard all trees where the PP is attached to that same constituent.

By using this approach, one does not need to examine each individual tree in the parse forest to get to the correct parse. A supporting workbench

makes available a set of discriminants, which are far fewer in number than the amount of trees in the parse forest. The human disambiguator can then go through this shorter list, marking the discriminants as valid or invalid, in order to gradually prune the forest down to a single parse.

The discriminant-based method of disambiguation can be further optimized by having the software automatically propagate decisions. For instance, when a particular analysis is discarded due to a manual choice by the human disambiguator, all discriminants that are true for that analysis can be automatically marked as being invalid. The practical upside of this is that, in most cases, only a subset of the discriminants needs to be manually marked in order to reduce the parse forest down to a single analysis, further reducing the amount of effort required.

For such an approach to work, it must be supported by a workbench that generates the discriminants and handles the pruning of the parse forest in a manner that is unobtrusive for the human annotator. For datasets in the HPSG family, this can be done using the `[incr tsdb()]` tool (Oepen and Flickinger, 1998). Besides providing an interface for the disambiguation process described above, this tool integrates functionality for benchmarking, profiling and testing the grammar over test suites.

4.2.1 Dynamic treebanks

The grammar-supported approach to treebanking can clearly reduce the amount of effort required to produce annotated sentences. It has another advantage that might not be obvious at first sight: It allows the development and maintenance of treebanks that are dynamic, in the sense described in (Oepen *et al.*, 2004) and (Branco *et al.*, 2009).

Even data that is already annotated may require an update to conform to new versions of the grammar. Having to re-annotate all of the existing corpora, even with the help of a grammar, becomes increasingly hard and, more than that, since changes to the grammar rarely invalidate previous analyses, much of the re-annotation work would ultimately result in picking the same parses as before.

The grammar-supported approach also mitigates this issue because, strictly speaking, it does not explicitly store the analysis picked by the annotator. Instead, it stores the individual discriminant decisions that lead to that analysis being chosen. Re-annotating a corpus with a new version of the grammar can then be as simple as automatically replaying those stored choices to obtain the correct analysis. Then, only the few sentences whose parse forests eventually do not contain the tree previously picked are brought to the attention of the human annotators.

4.3 CINTIL DeepBank

The dataset used in this work is CINTIL DeepBank (Branco *et al.*, 2010). It is being developed according to the process outlined in the previous Section by using the LX-Gram deep linguistic grammar (Branco and Costa, 2008, 2010) to automatically provide the parse forest and `[incr tsdb()]` to support the manual discriminant-based disambiguation process.

It is worth of note that, for this dataset, we have adopted a method of double-blind disambiguation followed by adjudication, widely adopted in the literature as the one that may best ensure the reliability of the data produced. In this setup, two human annotators work independently pruning the parse forest returned by the grammar. If both annotators agree on the choice of an analysis, that analysis is added to the dataset. When the annotators disagree on what is the preferred analysis, a third human annotator, the adjudicator, is brought in to decide which analysis will be added to the dataset, if any (the adjudicator is free to choose a third analysis, rejecting the ones chosen by either annotator).

Having more than one annotator working over the same data allows the calculation of inter-annotator agreement (ITA) metrics, which provide a measure of the consistency of the annotation process and of the reliability of the resulting dataset, with values above 0.8 considered to be reliable.

The ITA metric that is used looks at inter-annotator agreement over individual discriminant decisions. However, due to the sheer number of discriminants involved in pruning the parse forest of a single sentence, and to the fact that some discriminants are automatically flagged by the supporting annotation tool, the classic ITA metric had to be adapted for this particular task. For CINTIL DeepBank, under this adapted metric, the annotators achieved an ITA score of 0.86 (Castro, 2011).

This method of corpus annotation is resource-consuming, both in terms of human effort (three people are needed) and in terms of time (an adjudication round is required), but it allows a stricter quality control of the dataset being produced.

Both the grammar and the dataset continue under active development, following the approach of dynamic treebanking described earlier. The figures reported here are relative to the current stable version (v3).

The text is a 5,422 sentence corpus containing sentences taken from newspaper articles (4,644, or 86%) and sentences used for regression testing¹ of the grammar (778, or 14%).

¹Sentences for regression testing are usually short and purposely built by the grammar developer to target specific linguistic phenomena. They are used to test if changes or


```
<s> Maria/PNM[B-PER] Vitória/PNM[I-PER]
tem/TER/V#pi-3s[0] razão/RAZÃO/CN#fs[0] ./PNT[0] </s>
```

Figure 4.2: Snippet of CINTIL Corpus

Due to the way it was built, the dataset only contains those sentences that the supporting grammar is able to parse. There are 52,193 tokens in the dataset, which gives an average sentence length of 9.6 tokens. The sentences for regression testing are typically shorter, with an average length of 7.1 tokens. In the newspaper portion of the dataset each sentence has on average 10.0 tokens.

4.3.1 The underlying CINTIL corpus

CINTIL DeepBank, regression sentences aside, was built by running LX-Gram over the sentences in CINTIL Corpus (Barreto *et al.*, 2006) and manually disambiguating its output.

The sentences in CINTIL Corpus were annotated with a variety of shallow morphosyntactic information by manually correcting the annotation produced by LX-Suite, a set of shallow processing tools for Portuguese (Silva, 2007; Nunes, 2007; Martins, 2008; Ferreira *et al.*, 2007). The annotation consists of paragraph and sentence boundaries, part-of-speech categories, lemmas² for words from the open classes, inflection features (e.g. gender and number), and information on named entities boundaries and type (i.e. whether the entity is a person, location, etc).

An example showing the annotation format used in CINTIL may be seen in Figure 4.2 (Eng.: Maria Vitória is right). Sentence boundaries are given by `<s>` markup tags and token boundaries by white spaces. The POS tag is appended to the token, separated by a slash (e.g. PNM for proper name). When applicable, the lemma is placed between the word and the POS tag, between slashes (e.g. TER as the lemma of `tem`). Inflection features, when applicable, are placed after the POS tag separated by a hash mark (e.g. `pi-3s` for present indicative, 3rd person singular). Lastly, information on named entities is placed between square brackets and appended to the token. It follows the IOB scheme used, for instance, in the CoNLL 2002 shared task for named-entity recognition (Tjong Kim Sang, 2002): B (for Begin) marks a token that begins a named-entity (i.e. its first token); I (for Inside)

extensions to the grammar “break” any previous functionality.

²The lemma, also called dictionary form, is the base, non-inflected form of the word. This is typically the infinitive for verbs and the masculine singular for nouns and adjectives, when available.

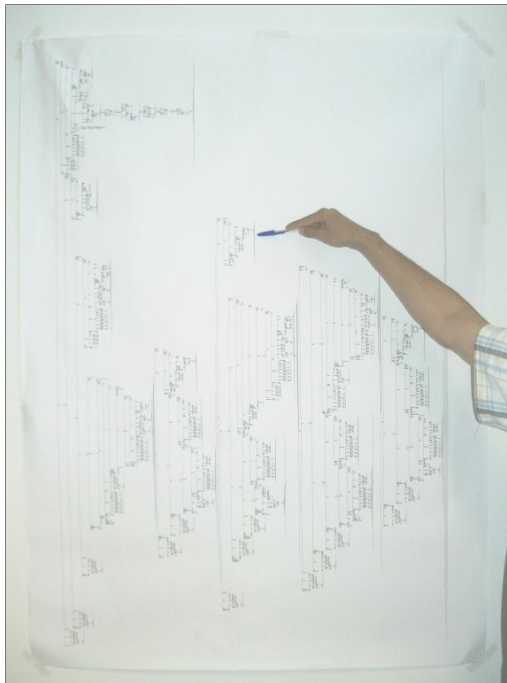


Figure 4.3: Example of a fully-fledged HPSG representation

marks tokens inside an entity (i.e. any token other than the first); and 0 (for Outside) marks tokens that do not belong to any entity. The type of the named entity is appended to the boundary mark (e.g. `PER` for person, `LOC` for location, `ORG` for organization, etc).

4.4 Extracting vistas

A concern that may be voiced regarding deep datasets like CINTIL DeepBank is that they are too theory-centric and that the linguistic information they store is found in a format that is too theory-specific and hard to access.

If, for instance, one only wishes to work with the constituency structure of the sentence, the full deep representation is too unwieldy. The image in Figure 4.3 helps to illustrate the problem. It shows the fully-fledged grammatical representation, under the HPSG framework, for the rather simple sentence *Todos os computadores têm um disco* (Eng.: All computers have a disk).³

³The printout is in a 6 pt font. The arm and hand holding a pen are there just to give a sense of the size of the grammatical representation.

Thus, it is desirable to have a procedure for extracting *vistas* (Silva *et al.*, 2012). That is, subsets of the information contained in the full dataset, such as text annotated only with part-of-speech tags, constituency trees or grammatical dependency graphs.

To address this, we developed a tool, `lkb2standard`, to extract normalized trees from the files exported by `[incr tsdb()]`, which are files that contain fully-fledged HPSG grammatical representations for the corresponding sentences.

Trees are used to represent constituency relations. The normalized trees produced by `lkb2standard` are decorated with extra information that is not related to constituency. Namely, grammatical dependency and semantic role tags are added to some phrasal nodes. Such extended treebanks are commonly called PropBanks.

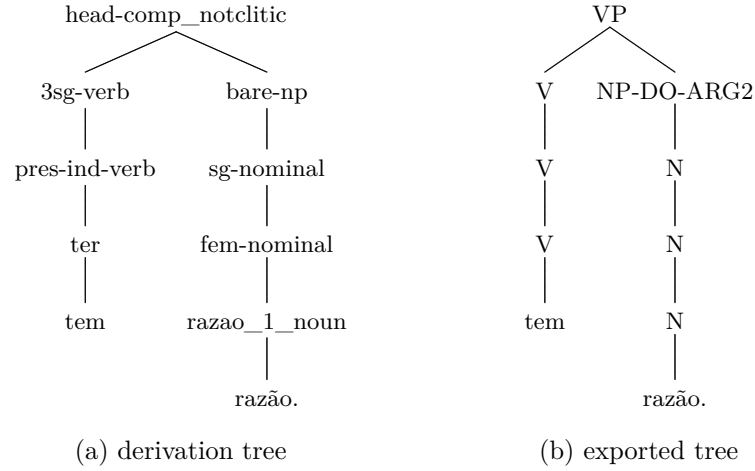
The vistas that are extracted are mainly used in two scenarios. On the one hand, as a resource for linguist studies, mainly oriented towards being looked at by a human. On the other hand, as a dataset for training and evaluating machine-learning algorithms. These different purposes lead to some differences in what is included in the vista. The discussion that follows will draw attention to these differences where they are relevant.

4.4.1 CINTIL PropBank

The PropBank is a dataset similar to what is described in (Kingsbury and Palmer, 2003) in that it consists of a layer of semantic role annotation that is added to phrases in the syntactic structure of the sentence. The format of these extended nodes in CINTIL PropBank is C-GF-SR, where C is the constituency tag, GF corresponds to the grammatical function and SR is the semantic role.

The first issue that arises when trying to obtain a standard representation for grammatical structures is that there is no such thing as an universally accepted normalized representation. This, however, is not the focus of the present work. For a detailed account of the design decisions that motivate the tree extraction, see (Branco *et al.*, 2011b).

The vista extraction procedure consists of several steps, each having to deal with non-trivial issues. As it turns out, many of these steps rely on features that are specific to LX-Gram (e.g. the names of certain grammar rules), which prevents the tool from being immediately applicable to the output of other deep grammars. Nevertheless, a description of the main steps, and of the problems that were found, will certainly be of help to others trying to tackle the same task for their grammars.

Figure 4.4: Derivation tree and exported tree from `[incr tsdb()]`

Retrieving the exported tree

The deep grammatical representation of a sentence that is exported by `[incr tsdb()]` at the end of the manual disambiguation process includes, alongside the full AVM, the derivation tree, which is a structure that encodes the rules that were used by the grammar during the analysis of that sentence and the order in which they were applied. The representation also includes a second tree which has the same structure as the derivation tree, but where the rule names have been replaced by syntactic categories through a mapping defined by the grammar creator. It is this second tree, which will be called *exported tree*, that is taken by `lkb2standard` as the starting point for the vista extraction procedure. Figure 4.4 shows these two trees. For the sake of readability, only a sentence snippet is shown.

Tokenization

Given that CINTIL Corpus contains information not present in the deep dataset that has been created by the grammar (e.g. information on the type of named entities), we want it to be possible to incorporate the data from the annotated sentences in CINTIL Corpus into the vistas extracted from CINTIL DeepBank. Moreover, due to the inner workings of `[incr tsdb()]`, the leaves in the trees are all converted to lowercase and truncated to the first 30 characters. The most straightforward way of fixing these issues is to replace each leaf by the corresponding token from the sentence.

For this procedures to work, leaves and tokens must be aligned. There is not a one-to-one correspondence between the leaves in the tree exported from

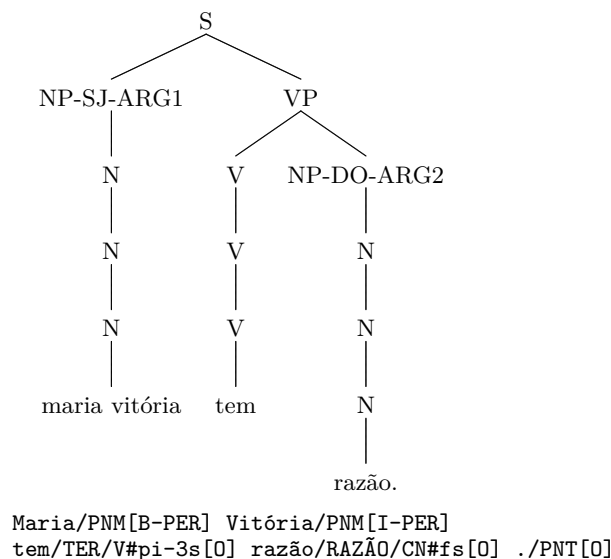


Figure 4.5: Exported tree and annotated sentence

the DeepBank and the tokens in the sentence in the corpus due to different criteria for tokenization used by the grammar and in the annotation of the corpus.

The reason for this misalignment between leafs and tokens is twofold: (i) the tree may have leafs for multi-word named entities that correspond to several tokens in the sentence; and (ii) in the tree, punctuation symbols are still attached to words, while they are found tokenized (i.e. detached from words) in the sentences of the corpus.

Figure 4.5 shows a short sentence (Eng.: Maria Vitória is right) from the corpus and, above it, the corresponding exported tree to better illustrate the mismatch between the number of leafs (3) in the tree and the number of tokens (5) in the sentence.

Multi-word named entities. Figure 4.6 shows an example of how multi-word named entity leafs are handled. Note that assigning constituency structure and correct categories to the words forming it is outside the scope of the `1kb2standard` tool. Accordingly, the entity in (a) is simply split into a set of nodes, one per word, all at the same depth and having the same category that the named entity had, as seen in (b).

When the vista is being extracted for the purpose of linguistic studies, the nodes are expanded only as a temporary transformation to allow aligning leafs and tokens and thus incorporate data from CINTIL into the treebank. After this information is merged into the treebank, the expansion is undone

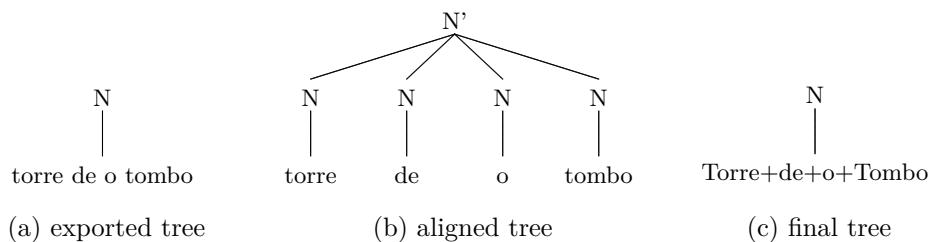


Figure 4.6: Multi-word named entities

so that the multi-word entity appears as a single node in the final normalized tree, shown in (c).

Punctuation. The purpose of the tokenization stage is only to obtain a one-to-one correspondence between the leafs in the tree and the tokens in the sentence. Accordingly, punctuation symbols are detached from words and placed in a newly created sister node. The process of moving the punctuation symbols to their correct position in the final normalized tree merits a slightly more detailed explanation and is addressed further ahead.

Feature bundles

With the leafs in the exported tree aligned one-to-one with the tokens in the sentence from the corpus, the tool can easily copy the information from those tokens over to the treebank as feature bundles that are appended to the leaf nodes.

Since one of the intended purposes for extracting the vistas is to use them to train a classifier that assigns deep lexical types, this information must also be extracted and included in the feature bundles. However, the deep lexical type of each word is not found in CINTIL, in the exported tree, or even in the derivation tree. The deep lexical type information in DeepBank can only be explicitly found in the full AVM representation, which is an unwieldy data structure.

A more practical way of accessing the deep lexical type of each word is through the derivation tree. The pre-terminal nodes of the derivation tree bear the name of the lexical rule that was applied to the word. The rule name can then be mapped to a lexical type by using the lexicon of the grammar.

Figure 4.7 shows an example taken from LX-Gram. The word *jantar* can be a noun (Eng.: dinner) or a verb (Eng.: dine) and, accordingly, has two separate entries in the lexicon. The pre-terminal node above the word

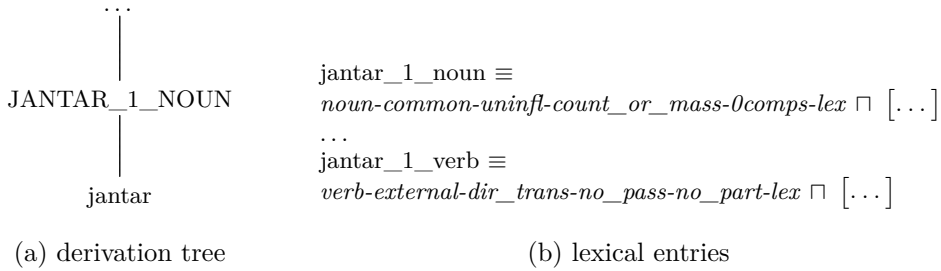


Figure 4.7: Mapping derivation rules to lexical types

`vinho/CN:VINHO:ms:0:noun-common-masc-0comps-lex`

Figure 4.8: A leaf with its full feature bundle

in the derivation tree indicates the grammar rule that was applied for that word which, in this example, is `JANTAR_1_NOUN`. The corresponding lexicon entry is then used to map the rule name into a lexical type (in this case, *noun-common-uninfl-count_or_mass-0comps-lex*).

With the full feature bundle, the leaves of the resulting normalized tree look like the example shown in Figure 4.8.

Note that the full feature bundles will not be shown in the remaining examples for the sake of readability.

Moving punctuation

Punctuation symbols were detached from words and placed in a temporary position. The present step is concerned with deciding where in the final normalized tree to place the node with the detached punctuation symbol since its final position will depend on the syntactic construction the symbol is a part of.

Coordination is represented as a recursive tree structure where several constituents of the same type are combined together. Usually, a comma is used to separate each constituent, except for the last one which is delimited by an explicit conjunction, such as *e* (Eng.: and), *ou* (Eng.: or), etc.

As the example in Figure 4.9 shows, the comma is initially attached to the final word in a constituent of the enumeration. After being detached from the word, it is placed in a new node (PNT) which is in an adjunction to the node to the right.

Appositions inside NPs are delimited by commas. This is made explicit in the tree by having the apposition in a sub-tree that is itself delimited by a pair of matching punctuation nodes. An example is shown in Figure 4.10.

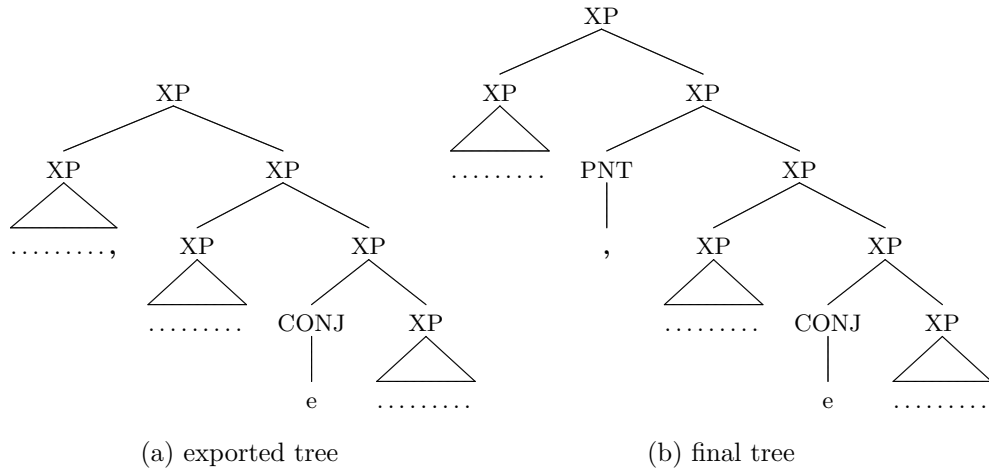


Figure 4.9: Coordination

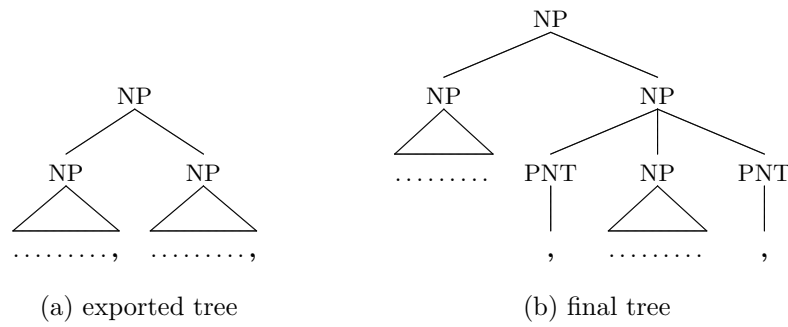


Figure 4.10: Apposition

Parenthetical structures and quoted expressions are represented in a similar way. These are also the only situations where ternary nodes are used.

In all other cases, such as for instance sentence-ending punctuation or topicalization, punctuation is raised as far as possible without crossing constituent boundaries.

Collapsing unary chains

In the exported tree, the syntactic representation contains chains of unary nodes with the same label. As mentioned above, this happens because the exported constituency tree mirrors the structure of the derivation tree. Each node in these chains corresponds to the application of a unary morphological rule (recall the example from Figure 4.4 and see (Copestake, 2002, Section 5.2) for more on such rules).

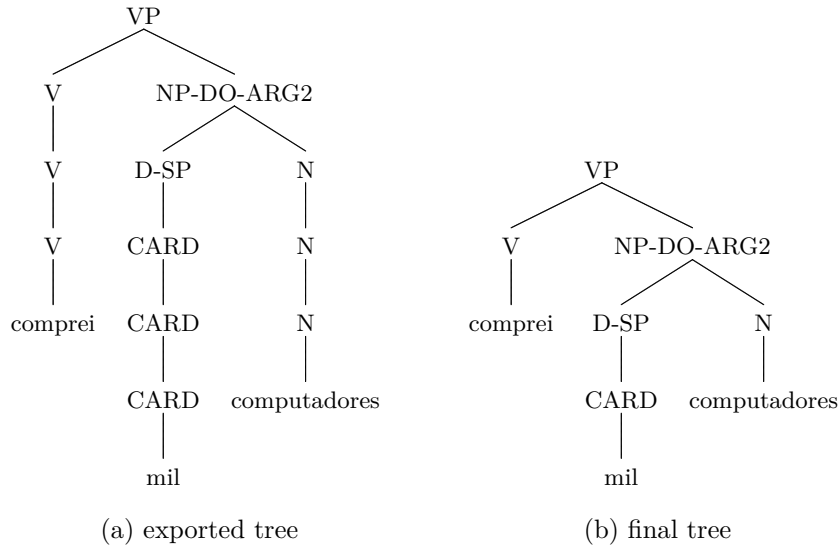


Figure 4.11: Unary chains

For instance, the unary chain of three N nodes that dominates the word *computadores* (Eng.: computers) in Figure 4.11(a) corresponds, from the bottom up, to the application of the following morphological rules: COMPUTADOR (the rule for the lexical entry of the word), MASC-NOMINAL (flags a feature that marks the word as having gender inflection) and PL-NOMINAL (flags a feature that marks the word as having number inflection).

The various nodes in these unary chains do not represent an actual difference in syntactic constituency, and are thus collapsed into a single node in the final tree, as seen in Figure 4.11(b).

Adding phonetically null items

For the purpose of linguistic studies, nodes corresponding to null subjects (*NULL*), null heads (*ELLIPSIS*), traces of constituents (*GAP*) and “tough” objects (*TOUGH*) are explicitly added to the final tree. There are a number of aspects concerning this step that are worth pointing out.

Pattern matching over the constituency tree is not enough to always detect nodes for phonetically null items. Instead, to do that, one must look at the derivation tree, since the relevant information can be found in the name of the rule used by the grammar.

At this stage of processing, the constituency tree and the derivation tree—which at the start of the extraction process are isomorphic—do not have matching structures any longer, since the constituency tree has been

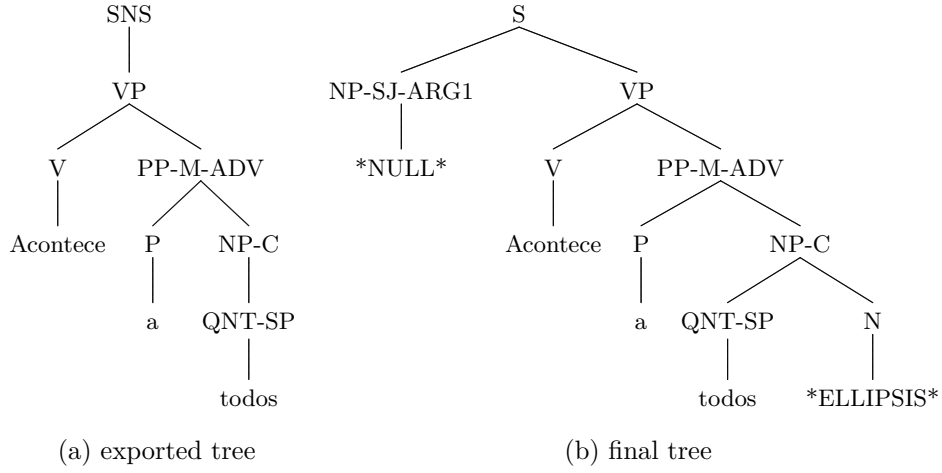


Figure 4.12: Null subjects and null heads

altered (viz. by moving punctuation and by collapsing unary chains). This issue was overcome by decorating the nodes in the exported constituency tree with the relevant information taken from the derivation tree while both structures are still isomorphic. In the previous pages and Figures, these node decorations have been omitted from the examples for the sake of clarity.

With the syntactic tree decorated, adding tree branches representing null subjects and null heads is quite straightforward.

Null subjects are found by looking for nodes in the constituency tree annotated with SNS tags, which are the way the grammar categorizes a sentence with a null subject. To properly assign a semantic role, the tool still needs to look at the corresponding node in the derivation tree, since the rule name indicates whether the missing NP-SJ node is an expletive (no semantic role), a passive construction (ARG2), a causative alternation (ARGA) or falls under the default case (ARG1).

Null heads are found by searching the derivation tree for certain rule names (or, more precisely, finding the relevant decorated node in the constituency tree). The rule name not only indicates the category of the missing head (i.e. whether it is nominal or verbal) but also whether it is the left or right child of the node.

Figure 4.12 shows an example of a parse tree with a null ARG1 subject and an elided nominal head as the right child of the NP-C node.

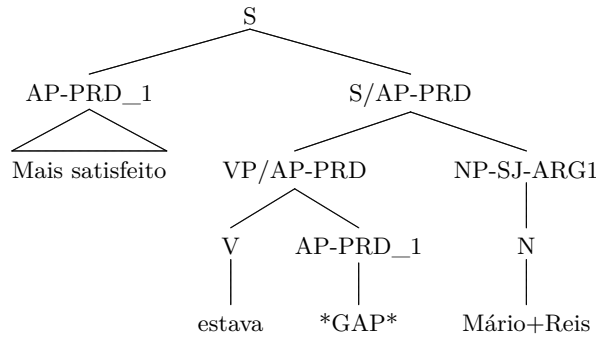


Figure 4.13: Traces and co-indexation

Nodes with a trace constituent are decorated by looking in the derivation tree for a rule that indicates the extraction of a constituent and marking the corresponding node in the constituency tree. The rule name also indicates whether the extracted constituent is on the left or on the right side of the node. The category of the extracted constituent is given by the usual HPSG slash notation, where a node labeled with X/Y indicates a constituent of type X with a missing constituent of type Y .

When adding the trace, it suffices searching for the decorated node and add the **GAP** node as its left or right child, depending on the marking. In addition, the trace is explicitly co-indexed with the displaced node by affixing the same index number to the trace and to the corresponding displaced constituent, as shown in Figure 4.13.

The displaced node, the AP-PRD *Mais satisfeito*, is found by following the path of slashed constituents from the trace up to the topmost slash, which is the sister node of the displaced node.

When the sister of the topmost slash is not of the expected category, indicated to the right of the slash, it signals a “tough” construction, and the trace node is marked with **TOUGH**, as shown in Figure 4.14.

We note again that information concerning phonetically null items is added only when the vista is to be used for linguistic studies. When the vista is to be used to train, say, a probabilistic parser, the nodes for phonetically null items are left out since they correspond, by definition, to items that will not overtly occur in the input of that parser.

Extending semantic role annotation

The semantic role information that is added to some nodes is at a different abstraction level than the constituency information conveyed by the tree

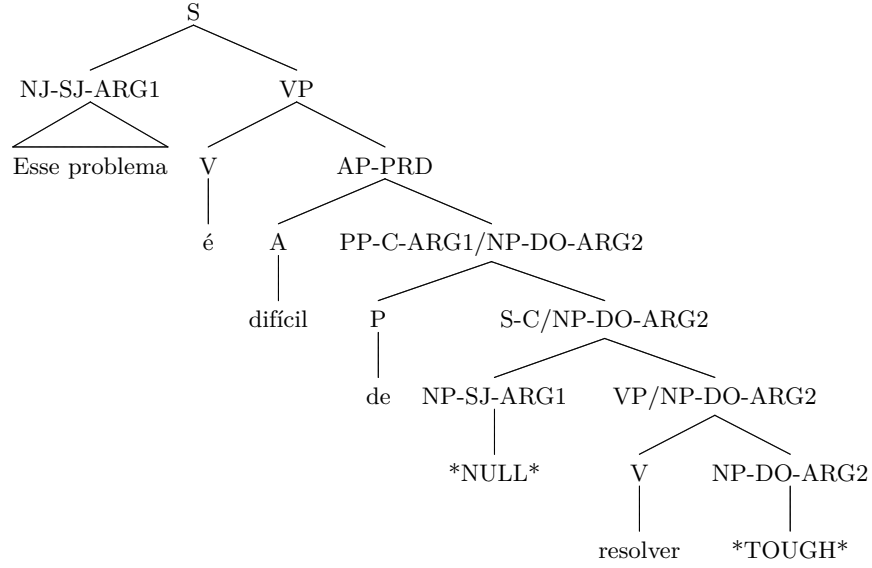


Figure 4.14: “Tough” constructions

structure. In particular, some roles are related to some constituent other than the one they are attached to.

This is the case in the context of complex predicates, such as modals, auxiliaries and raising verbs. In such cases, the semantic role tag of the subject node is suffixed with “cp” (for complex predicate). Anticausative constructions are handled in a similar way, but using “ac” as a suffix to the role tag.

For instance, the tree snippet in Figure 4.15 indicates that, though the NP is the subject of the topmost VP, it is the ARG1 of the head verb of the complex predicate *poderão começar*.

Arguments of control verbs are handled in a similar yet not fully identical manner. In LX-Gram the information that allows one to determine whether the verb at hand is a subject, direct object or indirect object control verb is found in the lexical type of the verb, and not on the derivation rule that it triggers. So, to get at this information, the grammar lexicon had to be used to map the derivation rule for the lexical entry of a word (i.e. the pre-terminal node in the derivation tree) into the corresponding lexical type.

Figure 4.16 shows an example, where the NP *As crianças* is the argument of the subject control verb *querem* and also of the embedded verb *dormir*.⁴

⁴The argument of a subject, direct object and indirect object control verb is marked, respectively, with ARG11, ARG21 and ARG31.

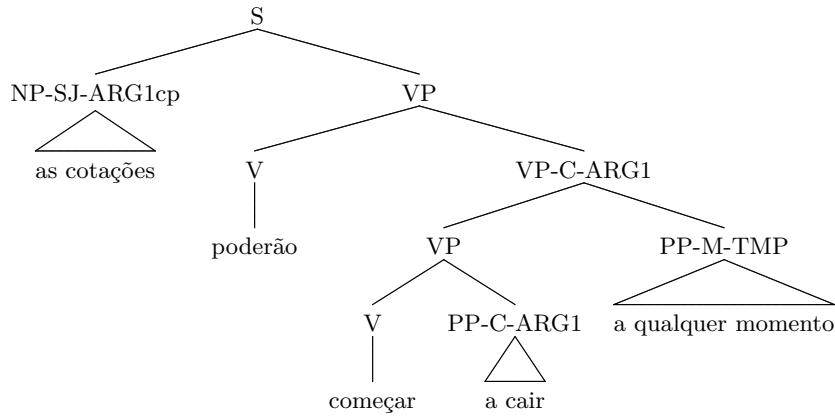


Figure 4.15: Complex predicate

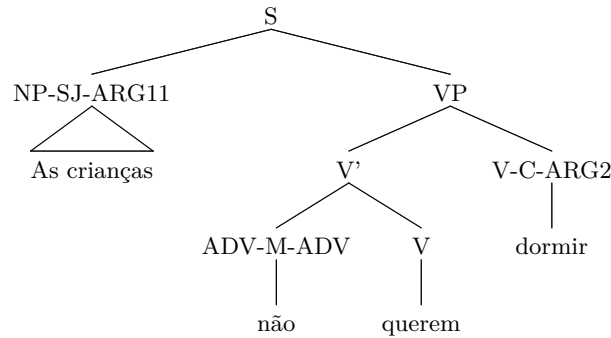


Figure 4.16: Control verbs

4.4.2 CINTIL TreeBank

Having extracted the PropBank vista, the TreeBank vista is straightforward to obtain by simply discarding all information on grammatical function and semantic roles, leaving only the lexical and phrasal constituency information in the nodes of the tree. That is, on each node annotated with a tag in the form C-GF-SR, the GF and SR fields are dropped, leaving only C.

4.4.3 CINTIL DependencyBank

Instead of giving a tree structure describing syntactic constituency, dependency parsing directly relates pairs of words by a grammatical function (i.e. SJ for subject, DO for direct object, M for modifier, etc).

It is important to note again that this information is expected to be particularly useful for the task of handling OOV words, since the dependents

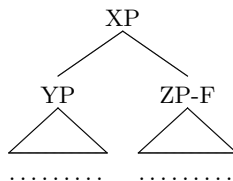


Figure 4.17: Extracting dependencies from PropBank

of a word are closely related to the SCF of that word.

The tool that extracts the dependency trees, `propbank2dependency`, was partly implemented by me (cf. Branco *et al.*, 2010). As such, for the sake of completeness, a quick overview of how it works is given here. For details on the design options concerning the grammatical dependencies and on the corresponding dataset, CINTIL DependencyBank, see (Branco *et al.*, 2011a).

Instead of going back to the DeepBank in order to extract the DependencyBank vista, we use the PropBank as an intermediate representation since it has already gone through an extensive normalization process. This is possible given that the normalized trees that form CINTIL PropBank have more than just information on syntactic constituency. The nodes in the trees also include information on grammatical function in tags that are attached to some constituency nodes (e.g. `SJ` for subject, `DO` for direct object, etc). This gives us a nice way to extract a dependency dataset from the PropBank.

Though there are a few cases that must be handled separately, the general case can be summarized as follows: Given that CINTIL PropBank adheres to an X-bar representation, phrasal nodes will have two children, one of which will be marked with a grammatical function. The (head of the) child that is marked is dependent on the (head of the) other child under that grammatical function. The head of the phrasal node is the head of the unmarked child.

For instance, the tree fragment shown in Figure 4.17 yields a dependency triple where ZP depends on YP under relation F. The head of XP is the head of YP.

This dataset thus produced is called CINTIL DependencyBank, and it follows the CoNLL format (Nivre *et al.*, 2007a): Sentences are separated by a blank line; one token per line, with several tab-separated fields, such as lemma, POS and head. Figure 4.18 shows the CoNLL representation of the sentence in Example (1).⁵

⁵Note that some CoNLL fields that are not relevant for the current task were omitted

id	form	lemma	pos	feat.	head	rel
1	A	—	DA	fs	2	SP
2	educação	EDUCAÇÃO	CN	fs	4	SJ-ARG1
3	não	—	ADV	—	4	M-ADV
4	tem	TER	V	pi-3s	0	ROOT
5	regras	REGRA	CN	fp	4	DO-ARG2
6	rígidas	RÍGIDO	ADJ	fp	5	M-PRED
7	.	—	PNT	—	4	PUNCT

Figure 4.18: CoNLL format (abridged)

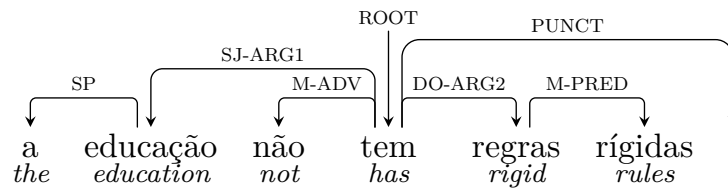


Figure 4.19: Dependency graph

- (1) A educação não tem regras ríidas .
 The education not has rules rigid .
 “Education does not have rigid rules.”

The lemma, POS and inflection features of each token are straightforward to obtain from the feature bundles in the PropBank. The last two fields encode the dependency relation for the token at hand. Namely, *head* is the *id* of the head token and *rel* is the name of the relation. Figure 4.19 shows these dependencies graphically.

4.5 Assessing dataset quality with parsers

One of the main concerns when designing a dataset is whether the chosen representation is consistent and provides the expected account of the various linguistic phenomena. However, this metric is hard to assess directly since there is no gold representation to compare against.⁶

An approach is to use the dataset as a resource, viz. as training data for a machine learning tool, and evaluate the performance of the resulting tool in comparison to similar tools trained over similar datasets. In a way, this resorts to performing an extrinsic evaluation of the dataset by taking the

from the example for the sake of clarity.

⁶Or, more precisely, it is that very same gold representation one is trying to obtain.

performance of a data-driven tool trained on that dataset as proxy for a direct assessment of dataset consistency and quality.

This Section reports on the training of several data-driven parsers over the vistas that were extracted.

4.5.1 Constituency parsers

Having extracted CINTIL TreeBank, the normalized constituency treebank vista described previously, probabilistic parsers were then induced over that dataset.

The initial experiment was done as part of a study (Silva *et al.*, 2010a) that assessed the suitability of available Portuguese treebanks and existing parser inducing tools for producing a state-of-the-art constituency parser for Portuguese. A guiding principle in that study was that the parsers should be created out-of-the-box. That is, the induced parsers should not be specifically tuned for Portuguese. The reason for this was twofold:

- Each probabilistic parser has a great deal of adjustable parameters, both for training and for parsing. Also, it is often possible to fine-tune the parser by adding code to handle language-specific cases (e.g. head-finding rules, heuristics for handling unknown words, etc). Fine-tuning would be done differently in each case, making it harder to perform a comparative assessment of parser inducing tools and resulting parsers. For instance, worse performance by any of the parsers could easily be caused by having a bad implementation of the language-specific tuning procedures, and not by the parser algorithm proper. Running the parsers out-of-the-box sidesteps these issues.
- It is to be expected that any kind of language-specific fine-tuning would improve—or at least not worsen—the performance of the parser. Running the parsers out-of-the-box is a quick way of getting a lower bound on the level of performance that it is possible to achieve.

There exist many freely-available software packages that in the past, and over several studies, have been used to induce state-of-the-art constituency parsers. Though they have been used mostly for English, the approaches are generally language-independent. After an overview of the existing packages, the following were found to be good candidates for inducing a parser for Portuguese:

Bikel (Bikel, 2002) is a head-driven statistical parsing engine. It is language-independent and highly configurable due to its use of language packages.

These are modules (Java code) that encapsulate all the methods that are specific to a language or treebank format.

Stanford (Klein and Manning, 2003) is a factored parser. This means that it resorts to separate models, one for phrase-structure and one for lexical dependencies, which are then factored together to generate the final tree. This parser also allows adding language-specific functionality by extending some of its (Java) classes.

Berkeley (Petrov *et al.*, 2006) automatically creates a base X-bar grammar from the training dataset through a binarization procedure that introduces branching X-bar nodes to ensure that every node has at most two children. This base grammar is then iteratively refined by splitting and merging non-terminal symbols.

Evaluation followed a standard 10-fold cross-validation methodology, iteratively training over a random 90% of the treebank and evaluating the resulting parser over the 10% not used for training, repeating this 10 times, and averaging the results. The following metrics were computed and averaged over every fold:

Parseval is a classic metric for bracketing correctness, widely used since its introduction in (Black *et al.*, 1991). A slight variant is used that also takes into account whether the constituent label is correct (Magerman, 1995). This metric provides labeled recall and labeled precision, which are rolled together into a single f-score value.⁷

Evalb is similar to Parseval, in that it is also a metric of bracketing correctness that provides labeled recall and labeled precision. The main difference between the two metrics is that Evalb counts terminal and pre-terminal nodes separately from the remaining phrasal nodes, which are referred to as the *roof* of the tree (Emms, 2008). This allows for a separate measure of part-of-speech (POS) accuracy, i.e. the accuracy in labeling pre-terminal nodes, while bracketing correctness refers only to the roof of the tree.

LeafAncestor is not so widely used as the two other metrics, but Sampson and Babarczy (2003) argue that it mirrors more closely our intuitive notions of parsing accuracy. Instead of measuring bracketing correctness, this metric checks the lineage—the sequence of constituents that connect a node to the root of the tree—of each terminal element. The

⁷F-score is the harmonic mean of precision and recall: $f = \frac{2pr}{p+r}$

4. DATASETS

	f_{Parseval}	f_{Evalb}	POS acc.	LeafAnc.
Bikel	85.16%	73.48%	90.26%	90.49%
Stanford	88.69%	79.63%	94.48%	92.06%
Berkeley	89.55%	81.60%	92.74%	94.16%

Table 4.1: Out-of-the-box constituency parser performance for v2

strings representing the lineages of a same token in two different parses can then be compared using a variant of string edit distance. The overall score is the average of the lineage scores.

Each parser package was used to induce a model over CINTIL TreeBank, as extracted from DeepBank by `1kb2standard`. At the time of the study reported in (Silva *et al.*, 2010a), the current stable version of CINTIL DeepBank (v3) was still under development, so the stable version at that time (v2), which contains 1,204 parses, was used instead.

Table 4.1 summarizes the results that were obtained.⁸ It shows the labeled Parseval f-score (f_{Parseval}), the separate bracketing and POS accuracy scores given by Evalb (f_{Evalb} and POS acc.), and the LeafAncestor score (LeafAnc.).

These values fall below what has been consistently achieved by the best parsers for English, which tend to score above 85% of bracketing correctness under the Evalb metric (Petrov *et al.*, 2006),⁹ but they were nevertheless quite encouraging because they were obtained with a very small dataset and with out-of-the-box parsers. Moreover, to the best of our knowledge, these were at the time the best published results for the probabilistic parsing of Portuguese (Silva *et al.*, 2010a,b).

The results obtained were thus a strong indicator that the treebank has a very high quality. This claim is given credence by the fact that Wing and Baldrige (2006) induced a finely-tuned Bikel parser over a different, larger (9,374 sentences) Portuguese treebank, and got a worse Parseval score of 63.2%. The fact that, in that study, the authors used a treebank that was manually annotated without thorough annotation guidelines and without

⁸In (Silva *et al.*, 2010a) several variants of the treebank are tested. The scores presented here correspond to the “NE-joined” variant, where multi-word named entities are represented as single leafs.

⁹The original Parseval metric has largely been deprecated. Current studies, when presenting a parseval score, are in fact using the Evalb roof bracketing correctness.

	f_{Parseval}	f_{Evalb}	POS acc.	LeafAnc.
Stanford	89.63%	80.36%	96.68%	92.27%
Berkeley	88.37%	81.69%	92.91%	93.26%

Table 4.2: Out-of-the-box constituency parser performance for v3

following double blind annotation with adjudication is a likely explanation for the lower performance.

With the more recent version (v3) of the dataset reaching 5,422 parses, the same tests were run to obtain updated scores, choosing only the two parsers that had achieved the best scores and maintaining the same experimental methodology. These are summarized in Table 4.2.

There is a slight across-the-board improvement, though it is not as large as one might expect. The likely explanation for this lies in the composition of the corpus. As mentioned in §4.3, part of the sentences in the corpus are used for regression testing. These amount to 14% of the v3 dataset, but formed a much larger relative portion of the corpus in the previous version. Since the sentences used in regression testing tend to be short, the v3 corpus can be seen as having proportionally fewer easy cases.

The results remain promising given that they were obtained using only out-of-the-box parsers and the current version of the treebank, though larger than the previous one, is still much smaller than, for instance, Penn Treebank, which is commonly used to obtain constituency parsers for English that fall in the 85%–90% range. As such, the results again argue for the high quality of the dataset.

4.5.2 Dependency parsers

The CINTIL DependencyBank vista can also be extrinsically evaluated by using available software packages that, in other studies, have shown to have the potential to create data-driven parsers with state-of-the-art performance. For this, the following tools were used:

MSTParser (McDonald *et al.*, 2005) is based on a two-stage method that starts by assigning an unlabeled dependency structure according to a large-margin discriminative model. In the second stage, a separate sequence classifier assigns labels to the dependencies.

4. DATASETS

	UAS	LAS
MSTParser	94.50%	88.59%
MaltParser	93.81%	87.54%

Table 4.3: Out-of-the-box dependency parser performance for v3

MaltParser (Nivre *et al.*, 2007b) is a generic parsing framework formed by three components: a parsing algorithm working over state transitions; a feature vector representation of the parser state; and a classifier that, given a parser state, chooses a parser action. The tool allows changing any of these three components.

Evaluation again follows a 10-fold cross-validation methodology, but in this case only two metrics are calculated: The unlabeled attachment score (UAS) gives the accuracy in determining that two words are related through a dependency relation, while the labeled attachment score (LAS) measures the same but also takes into account whether the label of the relation type was correctly assigned. The latter is, naturally, always equal or lower to the former.

Each parser package was run out-of-the-box over the 5,422 sentences in the extracted CINTIL DependencyBank vista. Table 4.3 summarizes the UAS and LAS scores that were obtained.

The results are quite good, in line with what has been reported for other languages, despite the small dataset and having placed no effort in fine-tuning the parsers.¹⁰

As with the constituency parsers from the preceding section, the results from this extrinsic evaluation using dependency parsers indicate that the dataset is of standard quality.

4.6 Summary

This Chapter covered the work that went into the preparation of the datasets that will support the training and evaluation of the classifiers for deep lexical types that will be presented next.

¹⁰MaltParser can be optimized with little effort through the use of the MaltOptimizer tool (Ballesteros and Nivre, 2012), which automatically runs and evaluates the parser under different settings. With this, MaltParser gets 94.74% UAS and 88.24% LAS, which are similar to the scores achieved by MSTParser.

After an overview of the two main representations of relevance, syntactic constituency and grammatical dependencies, the grammar-supported method of treebanking was introduced. This method allows the creation of linguistically rich, accurate and consistent datasets, of which CINTIL DeepBank is an example.

Such deep datasets are highly complex and, for many applications, the representation is too unwieldy. This motivated the creation of a tool for extracting *vistas*, i.e. subsets of the information contained in those datasets. With this tool, a “standartized” X-bar vista was obtained, CINTIL PropBank, from which, in turn, two additional vistas were extracted. One for syntactic constituency, CINTIL TreeBank, and the other for grammatical dependencies, CINTIL DependencyBank.

The quality of these extracted vistas was evaluated extrinsically, by inducing probabilistic parsers over them. These parsers obtained very good results, on a par with state-of-the-art scores reported for other languages, a fact that argues for the reliability of the datasets.

Chapter 5

Deep Lexical Ambiguity Resolution

This Chapter covers the experiments carried out in order to obtain and intrinsically evaluate classifiers that assign deep lexical types. It starts with a few remarks on the evaluation methodology that is used throughout the work. The following Section reports on initial exploratory experiments whose results guided subsequent research. The subsequent two Sections, §5.3 and §5.4, present and evaluate the approaches that were followed to build the classifiers. These are then further evaluated, over automatically predicted features in §5.5, over extended datasets in §5.6 and, in §5.7, compared against using the disambiguation module of the grammar itself. Finally, in §5.8 the classifiers are used also with an English grammar and dataset.

5.1 Evaluation methodology

When choosing an evaluation methodology, an important concern is to ensure that the datasets used for training and for evaluation are disjoint to allow measuring how well the model that was induced generalizes to previously unseen data.

For large-enough corpora, one can usually afford to simply put aside a portion of the data for testing and use the remainder for training. This is the usual approach with studies that use, for instance, the Penn Treebank, where,

for the sake of comparability, the same section of the corpus is typically used for evaluation by different studies. When the corpus is not so large, as it happens here with the CINTIL DeepBank, a major concern is making the most of the available data. The usual way of pursuing this is through n -fold cross-validation.

We opted for the common approach of taking $n = 10$, where the corpus is split into 10 equally-sized disjoint parts, the folds. Each fold is used to evaluate a classifier trained over the remaining 90% of the corpus and the 10 individual results are averaged. In this way, not only is each training phase able to use most (90%) of the data, as the totality of the corpus ends up being used for evaluation. All this while maintaining a separation between the data used for training and for evaluation.

The remaining decision then hinges on how to split the folds, which can be, for instance, (i) contiguous portions or (ii) randomly selected. The dataset is composed of a set of sentences created for regression testing of the grammar, followed by sentences from newspaper articles. Given this sharp divide between sentence types in the dataset, a random selection of sentences is used to form the splits. More precisely, the sentences in the dataset are first randomly shuffled, spreading the various sentence types uniformly over the corpus, and then 10 contiguous folds are taken.

5.2 Preliminary experiments

This Section reports on a set of initial experiments that were carried out using the previous stable version (v2) of CINTIL DeepBank.

During the course of the dissertation, a new and larger stable version of the dataset became available, and the experiments reported in the Sections that follow this one will be done over this new corpus. Nevertheless, it is important to report on these preliminary experiments since they were important in providing an initial direction and a compass that guided the approaches that were subsequently followed.

The preliminary experiments were in part prompted and inspired by some of the related work covered in Chapter 2. Following Dridan (2009, §5), two supertaggers that assign tags sequentially were induced, one using TnT and another using C&C. Next, a dedicated instance classifier is created, with a setup similar to the one presented in (Baldwin, 2005). The goal was, on the one hand, to replicate these previous approaches, and on the other hand, to study if and how the different dataset might have an impact in our expectations.

5.2.1 Preliminary sequential taggers

The task of assigning lexical types to tokens can be envisaged as a POS tagging task with an unusually detailed tagset. Hence, the most straightforward way to quickly create a supertagger is to train a POS tagger over an annotated corpus where POS tags have been replaced by lexical types.

POS tagging has been one of the most widely and intensely studied tasks in the field of NLP. Accordingly, many POS taggers have been created and made available over the years. Though there is a variety of approaches, those that became popular achieve roughly the same scores, indicating that a performance ceiling has likely been reached by the different machine learning approaches to this task.¹

For this experiment, we opted for the TnT POS tagger. This is a statistical POS tagger, well known for its efficiency in terms of training and tagging speed, and for achieving state-of-the-art results despite having a quite simple underlying model (Brants, 2000). It is based on a second-order hidden Markov Model² extended with linear smoothing of parameters to address data-sparseness issues and suffix analysis for handling unknown words.

Running with default parameters, TnT was trained over the 1,204 sentences extracted from CINTIL DeepBank (v2), with an overall tagset with 274 deep lexical types. To mitigate data-sparseness issues, words were replaced by their lemmas. The evaluation methodology followed was 10-fold cross-validation over random 90%/10% splits. The average token accuracy was 88.58%.

The accuracy achieved in this initial experiment is not far from the results reported in (Dridan, 2009), where a token accuracy of 91% was achieved, also when using TnT to assign lexical types, though with a more extensive tagset and larger training corpus (cf. §2.2.1, page 26).

C&C is a supertagger with a richer model than that used by TnT. In particular, its model is based on a Maximum Entropy approach and uses words and POS tags from a five word window as features (Clark and Curran, 2003, 2004, 2007).

Running with default parameters, the C&C supertagger was trained over the 1,204 POS-tagged sentences extracted from CINTIL DeepBank (v2).

¹This might not be exactly true. Banko and Brill (2001a,b) find, albeit for a simpler task, that learning curves can still grow as corpora increases exponentially in size into the billion of tokens. However, it is unlikely such a POS-tagged corpus could be created.

²See (Manning and Schütze, 1999) for a comprehensive coverage of this subject.

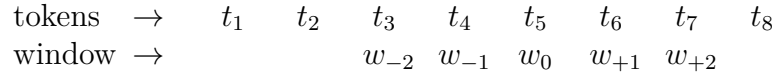


Figure 5.1: The 5-word window of C&C on an 8-token sentence

Similar to what was done with TnT, words were replaced by their lemmas to reduce data-sparseness.

The evaluation methodology was 10-fold cross-validation, taking the same 90%/10% splits as before. The gold POS tags were kept in the evaluation fold since C&C uses them as features and the purpose of the test was to intrinsically evaluate the classifier, not considering mistakes that might be introduced by a POS tagging step. The average token accuracy was 79.61%.

The C&C supertagger displayed a lower accuracy than TnT. This result is worthy of being addressed in further detail.

One would expect that C&C would perform better given that it uses a more complex model than TnT. While it is true that more complex models tend to be more accurate, they also lead to additional data-sparseness issues since more data is required to estimate the parameters of the model. Given that the experiments were run over a modestly sized dataset, there was likely not enough data for C&C to induce a good model.

This hypothesis is supported by the results in (Dridan, 2009). Therein, TnT is able to score slightly better than C&C. However, and, more importantly, this author notes that, upon reaching a certain dataset size, the learning curve for TnT has flattened while the learning curve of C&C is still rising. This is an indication that C&C might eventually surpass TnT given enough training data.

An additional contribution to the low score of C&C is likely to come from the short average sentence length in the current version of the dataset. Given that C&C uses features from a 5-word window and the average sentence length in the dataset is 8 words, the context window is only fully usable for roughly half of the tokens in an sentence. As shown in Figure 5.1, the context window only fully covers tokens in the sentence when centered on one of the tokens from t_3 to t_6 .

5.2.2 Preliminary instance classifier

TnT and C&C can be seen as sequential classifiers since both go through each token in a sentence, assigning a tag to each of them. An alternative way of tackling the assignment of lexical types is to use an instance classifier.

description	#
a 3-word lemma-POS window	$3 \times 2 = 6$
the lemma-POS of the head	$1 \times 2 = 2$
up to 6 dependency triples	$6 \times 3 = 18$
total features	26

Table 5.1: Features for the TiMBL instance classifier

That is, a classifier that can be invoked to assign a lexical type to the token at hand.

Following (Baldwin, 2005), we opted for a memory-based learning (MBL) approach³ to build this classifier. In MBL, the training examples are stored and new instances are classified according to the examples they are closest to (under some metric).

To train the classifier we used TiMBL (Daelemans *et al.*, 2009), an efficient software package for MBL that has been widely used for NLP tasks. Given that TiMBL is a generic framework for MBL, it is up to the user to define and extract the context features that are to be used in each specific task.

The dataset used for this is CINTIL DependencyBank, after being extended with an extra column with the lexical type of the token. A tool was created to extract features from this dataset. The experiment reported uses the 26 features summarized in Table 5.1.

Like with the sequential classifiers, lemmas were used instead of words. The features are the lemma-POS pairs from a 3-word context centered on the token at stake; the lemma-POS pair of the head (i.e. the word that the token is dependent on); and up to 6 triples representing the dependents of the token, formed by the relation name, and the lemma and POS of the dependent.

The evaluation methodology was again 10-fold cross-validation, with the same 90%/10% splits as before. The instance classifier achieved an average token accuracy of 79.38%, similar to the score of the C&C supertagger.

Specialized binary classifier

Any classifier has to face data-sparseness issues. Instead of inducing a single classifier capable of assigning all types, a common approach to tackling this problem is to induce a set of independent, specialized binary classifiers,

³Also known as instance-based, case-based or example-based learning approach.

one for each type (Galar *et al.*, 2011). Each binary classifier is tasked with assigning a TRUE/FALSE distinction to each token, depending on whether that token is or is not a member of the class the classifier recognizes.

To experiment with this approach, a binary classifier was induced. For this experiment, instead of creating multiple classifiers, we focused on a single type, **verb-dir_trans-lex** (direct transitive verb), one of the most common verbal types.

Before proceeding, it is important to note that such a classifier cannot be properly evaluated using the accuracy metric. From the 9,789 tokens in the dataset, only 108 have the type **verb-dir_trans-lex**. Accordingly, a naive classifier that always assigns the FALSE class would have an accuracy of nearly 99%. When dealing with binary classifiers, and specially when there is a great deal of class skewness, a more informative metric is Area Under Curve, or AUC (Fawcett, 2006).

For the **verb-dir_trans-lex** type alone, the multi-class instance classifier scored 0.79 AUC. This increases to 0.80 AUC with the specialized binary classifier.⁴

Dealing with an unbalanced dataset

The problem of unbalanced datasets is usually tackled either by under-sampling the majority class or by over-sampling the minority class.

An interesting approach that is worth exploring is SMOTE, or Synthetic Minority Over-sampling TEchnique (Chawla *et al.*, 2002). If we visualize the instances as being on a n -dimensional feature space, this method effectively over-samples the minority class by creating new synthetic samples along the lines connecting minority examples to their nearest minority class neighbors.

The SMOTE implementation included in the Weka machine-learning package (Hall *et al.*, 2009) was used. Run with default parameters, it doubled the number of **verb-dir_trans-lex** examples, from 108 to 216. Following the usual evaluation methodology on this enlarged dataset, the specialized binary classifier achieves 0.91 AUC.

The big increase in AUC should be taken with a grain of salt, though. SMOTE is concerned only with creating new synthetic minority examples in order to make the decision area for the minority class more general. Given that it looks at feature-space rather than input-space, it is blind to application-specific issues and constraints. In particular, the newly created examples are not necessarily linguistically sound. For instance, one of the synthetic examples created by SMOTE had a punctuation token with a

⁴An AUC score will always be between 0 and 1. Note that a score of 0.5 corresponds to random choice. No realistic classifier will have a score below that value.

	accuracy
TnT	88.58
C&C	79.61
TiMBL	79.38

Table 5.2: Accuracy (%) results from the preliminary experiment

verb POS feature. This raised some fears of over-fitting the classifier to the particular dataset being used.

5.2.3 Summary of the preliminary experiments

This Section presented exploratory experiments.

Several machine learning methods were used to induce classifiers that assign deep lexical types. Two sequential classifiers were induced, one based on the TnT POS tagger, with 88.58% accuracy, and the other on the C&C supertagger, with 79.61% accuracy. The simpler model achieves a better score, likely due to data-sparseness issues raised by the small dataset and to C&C being run out-of-the-box. An MBL-based instance classifier was also induced that achieves an accuracy score of 79.38%, close to that of the C&C supertagger. These results are summarized in Table 5.2.

When presented with a problem with multiple classes, a common approach to try to improve accuracy is to replace a single multi-class classifier by several class-specific classifiers. An experiment was run to test a binary classifier specific to a lexical type (viz. `verb-dir_trans-lex`), but it showed only a negligible improvement over the performance of the multi-class classifier for that particular type. Finally, in an attempt to balance the highly skewed dataset used to train the binary classifier, the SMOTE over-sampling technique was applied in order to create new minority class examples. This did lead to a marked improvement, raising AUC from 0.80 to 0.91 for that single type being targeted, but the extent of its applicability was called into question since the newly created examples were not linguistically well formed.

5.3 Sequential supertaggers

This Section reports on a set of experiments that take the same approach of sequential supertagging presented in the preliminary experiment, but over a larger dataset, v3 of CINTIL DeepBank, that eventually became available.

5. DEEP LEXICAL AMBIGUITY RESOLUTION

accuracy		accuracy	
TnT	90.74	TnT	89.19
C&C	81.59	C&C	78.51
SVMTool	90.61	SVMTool	89.04
(a) over all tokens		(b) excluding punctuation	

Table 5.3: Accuracy (%) of sequential supertaggers

We keep the same taggers as in the preliminary experiment, TnT and C&C, and add SVMTool.

TnT (Brants, 2000) was used in the preliminary experiment (cf. §5.2.1) where it achieved the best performance among the approaches that were tried. As such, it will be used again for the experiments over the larger dataset.

C&C (Clark and Curran, 2003, 2004, 2007) was also used in the preliminary experiment (cf. §5.2.1), where it fared worse than TnT, perhaps partly due to data-sparseness issues. It is tested again to determine the impact of additional training data.

SVMTool (Giménez and Màrquez, 2004) is a POS tagger generator based on support-vector machines. It is extremely flexible in allowing to define which features should be used in the model (e.g. size of context window, number of POS bigrams, etc.) and the tagging strategy (left to right, bidirectional, number of passes, etc). Here it is run with the default and simplest settings, “M0 LR”, which use Model 0 in a left to right tagging direction.⁵ It is used since in (Giménez and Màrquez, 2004) achieves better results than TnT.

These taggers were run out-of-the-box and evaluated using 10-fold cross-validation, yielding the results summarized in Table 5.3. Accuracy is shown for all tokens and also for all tokens except punctuation, since tokens in this category are not ambiguous and thus inflate the accuracy score.

C&C again falls behind TnT, maintaining the same ratio difference in their scores as in the preliminary experiment (i.e. performance has improved for both, but C&C remains 90% as accurate as TnT).

⁵Model 0 uses features such as the last two POS tags, bigrams and trigrams of POS and words, and word affixes. See (Giménez and Màrquez, 2006) for a detailed list.

TnT is now close to the 91% token accuracy that was achieved by Dridan (2009) with that same tagger but over a corpus of English (cf. §2.2.1, page 26). As such, we intend to use TnT—and also SVM-TK, since their performances are similar—as representatives of sequential tagging approaches when comparing these approaches with the instance classifier, which is presented next.

5.4 Instance classifier

Most of the classifiers presented so far look at features that have been extracted from what basically amounts to a limited window of context around the word being tagged. However, in some cases, it might happen that some of the information that is relevant towards determining the lexical type of a word is not found within a window of context of predetermined size around that word.

In particular, given that the subcategorization frame (SCF) is one of the most relevant pieces of information that is associated with a word in the lexicon of a deep grammar, one would expect that features describing the inter-word dependencies in a sentence would be highly discriminative and help to accurately assign lexical types. However, these dependencies are often unbounded, and thus cannot be generally accounted for by features from a limited window of context.

The instance classifier based on TiMBL that was presented in the preliminary experiment (cf. §5.2.2) was a first step towards addressing this by including features for grammatical dependencies. That early experiment did not produce any improvement in accuracy when compared with the sequential supertaggers that were based on a limited window of context, though this might be due to the very small corpus of 1,204 sentences used at the time which did not have enough data to support the more complex features.

The preliminary experiment also highlighted the difficulty in designing and extracting features. The set of 26 features that was shown in Table 5.1 is just one of the many that were tested, and the 18 features representing dependencies were bounded at 6 triples because that is the highest number of dependencies connected to a single word that were found in the corpus used at the time. On the one hand, this means that for most cases many of these triples will be empty but, on the other hand, if a new case occurs that happens to require more than 6 dependencies, unlikely as it may be, the chosen features will not be able to account for all of them.

A better approach would be to use a classifier that somehow is able to

directly take a dependency graph as a feature. Support-vector machine algorithms, combined with tree kernels (cf. §3.2), provide a straightforward way of doing this, a consideration that lead to moving away from the TiMBL classifier from the preliminary experiment and to the development of the SVM-TK classifier presented next.

5.4.1 SVM-TK

As described in §3.2, tree kernels provide a way for a classifier to seamlessly use features based on tree structures.

For training and classification we use SVM-light-TK (Moschitti, 2006), an extension to the widely-used SVM-light (Joachims, 1999) software for SVMs that adds an implementation of a tree kernel function. This extension allows providing one or more tree structures directly as data (using the standard representation of trees using parenthesis) together with the numeric vector data that are already accepted by SVM-light.

Binarization

Given that the task at stake is a multi-class classification problem but an SVM is a binary classifier, the problem must first be binarized (Galar *et al.*, 2011), a technique that decomposes the original multi-class problem into several binary problems. This decomposition is usually done according to one of two strategies, one-vs-all (OvA) or one-vs-one (OvO).

When following the OvA strategy, one creates as many binary classifiers as there are classes, and each classifier is tasked with identifying instances that belong to its class.⁶ When inducing a model for a class, instances of that class are seen as positive examples while all other instances in the dataset are taken as negative examples. When classifying a new instance, ideally all but one classifier will respond negatively, and the class that ends up being assigned to that instance is the class that corresponds to the single classifier that answered positively. Cases where several classifiers answer positively must undergo some sort of tie-break procedure, which could, for instance, be based on the confidence score of the conflicting classifiers.

The OvO strategy consists of creating one binary classifier for each possible pairing of different classes, where each classifier is tasked with assigning one of the two classes it knows to the instances being annotated. This strategy decomposes a problem with n classes into $\binom{n}{2} = \frac{n(n-1)}{2}$ separate

⁶This was the approach taken in the preliminary experiment where a specialized binary classifier was created for `verb-dir_trans-lex`.

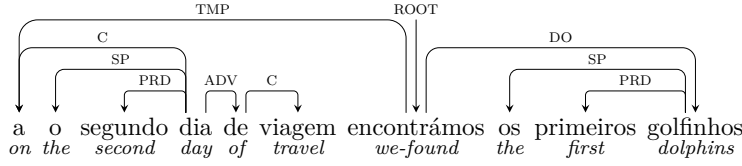


Figure 5.2: Dependency graph

binary problems (i.e. the binomial coefficient, n choose k , or one for each way of picking k unordered outcomes from n possibilities).

When inducing an OvO classifier that discriminates between class A and class B , instances of class A are taken as positive examples while instances of class B are seen as negative examples. When classifying, each classifier performs a binary decision, voting for one of the two classes it is tasked with discriminating between, and the class with the overall largest number of votes is chosen.

OvO creates many classifiers, which raises some concerns regarding its scalability when the number of classes is high, as it is bound to happen when wanting to assign deep lexical types. However, the training phase of an OvO classifier for classes A and B only looks at instances that belong to either of those classes, while each OvA classifier must be trained over every instance in the dataset. This greatly offsets the penalty of having such a large number of classifiers when adopting the OvO strategy.

For this reason, and because OvO seems to generally outperform OvA (Galar *et al.*, 2011), the OvO strategy is used in the current work.

Representing dependencies

The example in Figure 5.2 shows the dependency representation of the (tokenized) sentence “a o segundo dia de viagem encontramos os primeiros golfinhos” (Eng.: on the second day of travel we found the first dolphins).⁷ Note that, in the dataset, each word is also annotated with its lexical type, POS tag and lemma, though this is not shown in the example for the sake of readability.

Take, for instance, the word “encontrámos”, whose lexical type in this particular occurrence is `verb-dir-trans-lex`, the type assigned to transitive verbs by LX-Gram. The OvO classifier tasked with recognizing this type (against some other type) will take this instance as a positive training

⁷Relations shown in the example: ADV (adverb), C (complement), DO (direct object), PRED (predicate), SP (specifier) and TMP (temporal modifier).

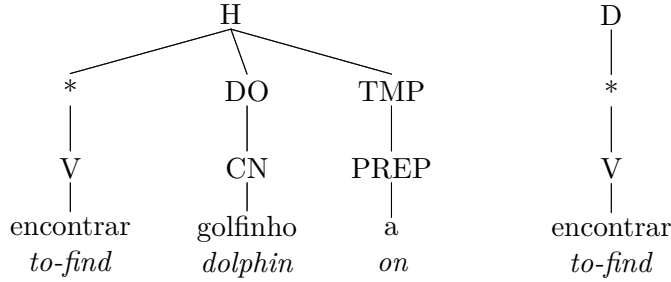


Figure 5.3: Trees used by SVM-TK for “encontrar”

example, while other OvO classifiers tasked with recognizing against this type would take this exact same instance as a negative training example.

The full dependency representation of the sentence has many grammatical relations that are irrelevant for learning how to classify this word. Instead, the classifier focuses more closely on the information that is directly relevant towards determining the SCF of the target word by looking only at its immediate neighbors in the dependency graph, viz. its dependents and the word it depends on. As before, words were replaced by their lemmas to reduce data-sparseness.

SVM-TK handles tree structures, and not generic graphs. Accordingly, the dependency information is encoded into the two tree structures shown in Figure 5.3, which represent the actual data given to SVM-light-TK.

One tree, labeled with H as root, holds the word and all its dependents. The target word is marked by being under an asterisk “category” while the dependents fall under a “category” corresponding to the relation between the target word and the dependent. The words appears as the leafs of the tree, with their POS tags as the pre-terminal nodes.⁸

The second feature tree, labeled with D as root, encodes the target word—again marked with an asterisk—and the word it is dependent on. In the example shown in Figure 5.3, since the target word is the main verb of the sentence, the D feature tree has no other nodes apart from that of the target word.

This is just one of the various ways that dependency information can be encoded into trees to be used by SVM-TK. It was chosen because, in early tests, the classifier showed slightly better performance when using it versus the alternatives. Nevertheless, the important point that must be underlined is that any alternative must also somehow encode the information one would expect to find in the SCF description of a word, corresponding

⁸POS tags in the example: V (verb), PREP (preposition) and CN (common noun).

in this example to the H tree: The target word is a verb, “encontrar”, it has as dependents a direct object with the common noun category, which in this case is “golfinho”, and the preposition “a” as a temporal modifier.

5.4.2 Restriction to the top- n types

The class distribution in the dataset is highly skewed. For instance, from all the common noun types that occur in the corpus, the two most frequent ones are enough to account for 57% of all the common noun tokens. Such skewed category distributions are usually a problematic issue for machine-learning approaches since the number of instances of the rarer categories is too small to properly estimate the parameters of the model.

Since SVM-TK is in fact composed by an ensemble of specialized OvO binary classifiers, for many types there are not enough instances in the dataset to train a classifier.

For example, there are 130 verb types in the corpus, but 13 of those are types whose occurrences are all found in a single fold. So, when that fold is the one being used for evaluation, none of the OvO classifiers that involve that type can be created, since there are no instances of that type in the remaining folds, which are the ones being used for training.⁹

Hence, the evaluation that follows is done only for the most frequent types (cf. Appendix A). For instance, top-10 means picking the 10 most frequent types in the corpus, training OvO classifiers for those types, and evaluating only over the tokens that in the original corpus bear one of those types. Though this is an apparently stringent restriction, the skewed distribution allows covering much of the corpus with only a few types. For instance, there are 130 verb types in the corpus, but the top- n most frequent ones are enough to account for most of the verb token occurrences in the corpus, namely top-10 covers 68%, top-20 covers 84% and top-30 covers 90% of verb occurrences.

Figure 5.4 shows a plot of the cumulative token coverage for the top- n most frequent verb types,¹⁰ highlighting the points where n is 10, 20 and 30. The halfway value, where $n = 65$, is also marked.

⁹In fact, only 32 of the 130 verb types occur in every fold. For each of the remaining 98 types, there is always some fold from where that type is missing. Half of the verb types occur 6 or fewer times in the corpus and only cover 3% of the verb tokens.

¹⁰The coverage curves for other open categories show a similar overall shape.

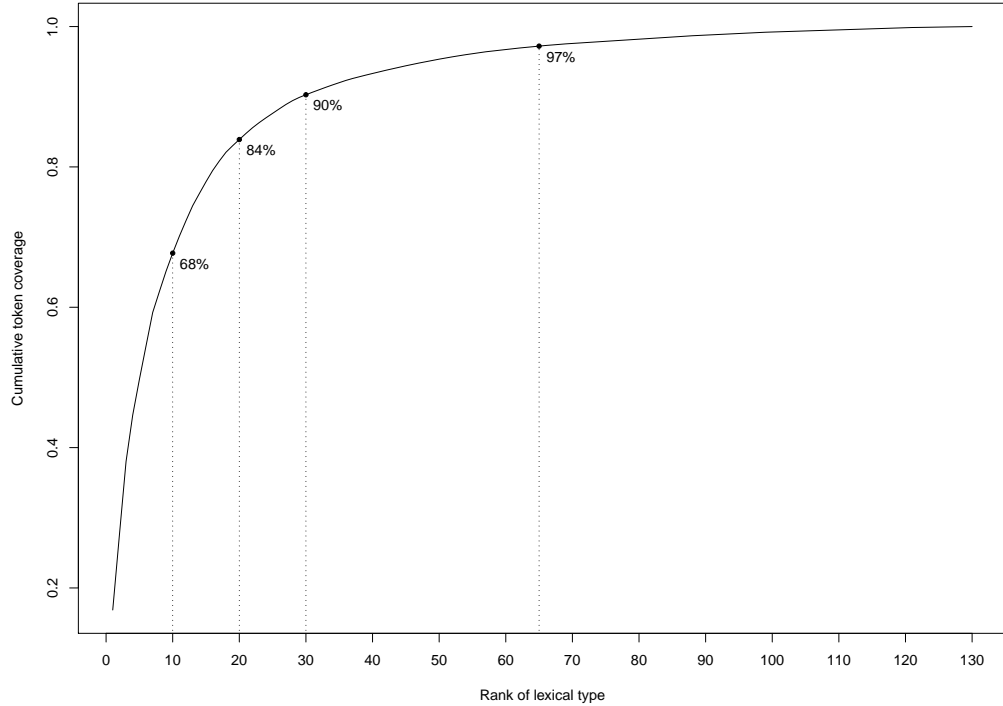


Figure 5.4: Cumulative verb token coverage of top- n verb lexical types

5.4.3 Initial evaluation and comparison

Varying the number of types that are assignable can be seen as in reality creating different classifiers. For instance, an SVM-TK classifier for the top-10 verb types might be very accurate over that particular set of types, but its accuracy over all verb tokens will always be limited by the low token coverage. That is, given that the top-10 verb types only cover 68% of the verb tokens, the remaining tokens will always receive the wrong type. A classifier induced for the top-30 verb types might have worse performance when measured over the target tokens but achieve a better overall score, since the verb token coverage, and thus the accuracy ceiling, is 90%.

This can be seen in Table 5.4, which shows the accuracy over verb tokens of the best sequential supertaggers presented previously and of three SVM-TK classifiers induced for three different top- n verb sets.

As n increases, the overall verb token accuracy of the SVM-TK classifier also increases due to the accuracy ceiling being raised. Although results here are only explicitly shown for top-10, top-20 and top-30, experiments found that, as expected, there are diminishing returns that come from increasing the value of n . Overall verb token accuracy stopped improving around the

accuracy	
TnT	90.19
SVMTool	90.94
SVM-TK	
for top-10	64.16
for top-20	75.74
for top-30	80.38

Table 5.4: Classifier accuracy (%) over all verb tokens

top-65 mark, where SVM-TK achieves 83.69% against a ceiling of 97%.

Regarding the sequential supertaggers, the results seem to indicate that the features used by SVMTool allow it to handle verbs better than the two other tools, since it is able to take first place from TnT, which scores slightly better over all tokens (compare Table 5.3 with Table 5.4).

The fact that SVM-TK has an unavoidable accuracy ceiling, which varies with the number of assignable types, makes it harder to interpret its performance scores. To provide an alternative view, accuracy can be measured only over tokens that in the gold corpus bear a type belonging to one of the top- n verb types.

For the SVM-TK classifier the training and evaluation procedure is rather straightforward, viz. induce a classifier for the top- n types and evaluate only over tokens that bear one of those types. For the sequential taggers, however, two approaches are possible:

- The taggers can be trained over a corpus that uses the full tagset of deep lexical types but evaluated only over the subset of tokens that in the gold corpus bear a type belonging to the top- n .
- Alternatively, the training data can be altered so that only the tokens with a type in the top- n have a fully realized deep lexical type, leaving all other tokens with a plain POS tag.

At first blush, the latter approach might seem better due to having a smaller, and thus easier, tagset. Since one is only going to evaluate over a restricted set of top- n types, there is no use in making the training tagset unnecessarily large. However, results have shown that the sequential taggers, even when being evaluated only over a top- n subset, perform better if trained over a dataset that uses the full tagset of deep lexical types. This is likely

	SVM-TK	TnT	SVMTool
top-10	94.76	92.96	94.20
top-20	90.27	91.69	92.49
top-30	89.04	91.62	92.48

Table 5.5: Classifier accuracy (%) over top- n verb types

due to the fact that the additional granularity in the context allows the taggers to be more discriminant.

Accordingly, in the results that follow, sequential taggers being evaluated over a top- n subset have been nevertheless trained over a corpus where every token bears a deep lexical type.

Table 5.5 summarizes the accuracy scores for the SVM-TK classifier and the two sequential taggers over different top- n subsets of verb types.

In all cases, accuracy decreases as the set of lexical types being considered is broadened by higher values of n . This is to be expected, since the most common types are tagged more accurately.

It is interesting to note that, for the top-10 most frequent types, SVM-TK is ahead, beating SVMTool and far surpassing TnT, but this advantage is quickly lost as the set of lexical types that SVM-TK must account for grows.

These results raise two related issues:

- SVM-TK starts off having better results, but that might happen only because it is using features that are based on gold dependencies instead of having to rely on automatically assigned grammatical dependencies for its features.
- The drop in accuracy shown by SVM-TK as n increases suggests the existence of a data-sparseness problem, which could be reduced given additional training data.

To address these two points, the classifier should be run using features that are extracted from automatically assigned dependencies and also trained and tested over a larger dataset. These issues are covered in the next Sections.

5.5 Running over predicted dependencies

The previous Section was concerned with evaluating the SVM-TK classifier alone. Accordingly, the features used by the classifier were taken from the gold dependencies in the corpus. However, on an actual running system, the

5.5. Running over predicted dependencies

	gold	pred.
top-10	94.76	93.88
top-20	90.27	89.34
top-30	89.04	87.99

Table 5.6: SVM-TK accuracy (%) over top- n verb types using predicted features

features used by the classifier must be based on the output generated by a dependency parser.

To test the performance under this scenario, we trained a dependency parser, MaltParser, that achieves an average 88.24% LAS.¹¹ This parser is used to automatically assign grammatical relations prior to running the SVM-TK classifier.

Note that the predicted dependencies produced by MaltParser are not used only when SVM-TK is assigning types, but also during the training phase of the classifier. This idea of using the output of a classifier (MaltParser) as training data for another (SVM-TK) has some similarities to the stacked learning approach presented in (Wolpert, 1992). By taking this approach, SVM-TK can be seen as learning to cope with the noisy features coming from MaltParser, which will be the actual input it will use, instead of training over unrealistic gold dependencies. In fact, experiments have shown that, by taking this approach, SVM-TK ultimately achieves slightly higher accuracy than when trained over gold dependencies and run over the output of MaltParser.

As expected, the noisy features derived from the automatically assigned dependencies have a detrimental effect on the accuracy of the classifier, but the decrease is in fact quite small. For the same top- n sets reported previously, the accuracy of the SVM-TK classifier when running over predicted dependencies trails 0.88–1.05 percentage points behind that of the classifier that is trained and run using gold dependencies, as shown in Table 5.6.

The detrimental effect the results from having automatically assigned dependencies is not dramatic. Note, however, that the most frequent verb types will also be the ones most likely to be annotated correctly. Accordingly, one would expect, and it turns out to be so, that as n is increased the accuracy gap between the classifier running over gold dependencies and the classifier running over predicted dependencies also increases. Over the top-65 verbs,

¹¹For this task, MaltOptimizer was used to automatically tune the settings (cf. §4.5.2).

dataset	sentences	tokens	unique	oov
base (v3)	5,422	51,483	8,815	10.0%
+ Wiki	10,727	139,330	18,899	7.6%
+ Público	15,108	205,585	24,063	6.6%
+ Folha	21,217	288,875	30,204	6.0%

Table 5.7: Cumulative size of datasets

for instance, the gap is 1.29 percentage points.

It is anticipated that using larger corpora will be effective in raising the accuracy of the classifier that runs over predicted dependencies since additional training data will not only improve the SVM-TK classifier itself but also the underlying parser that provides the dependencies that are used as features.

5.6 Experiments over extended datasets

The drop in accuracy of SVM-TK when the set of assignable verbs grows in size suggests a data-sparseness problem. To assess the impact of using additional training data we extend the dataset with automatically annotated sentences.

The extended datasets were created by taking additional sentences from the Público newspaper, as well as sentences from the Portuguese Wikipedia and from the Folha de São Paulo newspaper, pre-processing them with a POS tagger, and running them through LX-Gram. Having an analysis given by the grammar, the vista extraction procedure described in §4.4 allows obtaining the grammatical dependencies that are then used in the training of the classifier.

The grammar, whose coverage currently stands at roughly 30%, allowed building progressively larger datasets as more data were added. The cumulative sizes of the resulting datasets are shown in Table 5.7, together with the number of unique¹² tokens and the “expected” ratio of OOV words, which was determined by taking the average of the OOV ratio of each of the 10 folds (i.e. words that occur in a fold but not in any of the other 9 folds).

Such an approach is only made possible because, as discussed in §1.5, LX-Gram is run over the output of a POS tagger and is able to use the

¹²The number of unique tokens, i.e. the number of tokens when not counting multiple occurrences, is often referred to as the number of *types*, but this term was not used to avoid confusion with its use to refer to deep lexical types.

5.6. Experiments over extended datasets

dataset	TnT	SVMTool	dataset	TnT	SVMTool
base (v3)	89.19	89.04	base (v3)	90.19	90.94
+ Wiki	90.12	90.50	+ Wiki	89.77	91.29
+ Público	90.83	91.32	+ Público	89.98	91.70
+ Folha	91.53	92.09	+ Folha	91.08	92.74

(a) all tokens (excl. punctuation) (b) verb tokens

Table 5.8: Accuracy (%) of sequential supertaggers on extended datasets

tags that were assigned to pick a generic (or default) deep lexical type for OOV words; and also because LX-Gram, like many other HPSG grammars, includes a stochastic disambiguation module that automatically chooses the most likely analysis among all those returned in the parse forest, instead of requiring a manual choice by a human annotator (Branco and Costa, 2010).

This disambiguation module will make mistakes, and in some cases will not pick the correct analysis. However, it is not clear how these wrong choices translate into errors in the lexical types that end up being assigned to the tokens. For instance, when faced with the rather common case of PP-attachment ambiguity, the disambiguation module may choose the wrong attachment, which will count as being a wrong analysis though most lexical types assigned to the words in the sentence may nonetheless be correct.

To assess this, the disambiguation module was tested over the base dataset, for which the correct parses are known. This test found that the grammar picks the correct parse in 44% of the cases. However, when considering only the correctness of the assignment of lexical types, the sentence picked by the grammar is fully correct in 68% of the cases.

Tables 5.8 and 5.9 summarize the results of the evaluation over the extended datasets. Each table presents the results from previous experiments over the base (v3) dataset and over the progressively larger extended datasets.

Table 5.8(a) complements Table 5.3(b) (page 84) by showing how the accuracy of the best sequential supertaggers over all tokens (excluding punctuation) changes as the dataset grows. Table 5.8(b) is similar, but accuracy is measured only over verb tokens, complementing the scores shown in the upper part of Table 5.4 (page 91).

Nor surprisingly, both taggers improve their accuracy as additional training data are used. More interesting is to note how SVMTool begins to pull away from TnT as the size of the dataset increases. Henceforth, we use only SVMTool to represent the sequential tagging approaches.

dataset	top-10		top-20		top-30	
	Tool	TK	Tool	TK	Tool	TK
base (v3)	94.20	93.88	92.49	89.34	92.48	87.99
+ Wiki	93.58	93.83	92.75	90.35	92.73	88.97
+ Público	93.55	93.95	93.08	91.29	93.16	90.21
+ Folha	94.07	94.55	93.91	92.26	94.01	91.50

Table 5.9: Accuracy (%) comparison between SVMTool and SVM-TK (with predicted features), on extended datasets

Table 5.9 complements the information in Tables 5.5 and 5.6. For each dataset, and for each of the three top- n sets of verbs, it shows the accuracy of SVMTool and SVM-TK, the latter running over predicted dependencies assigned by MaltParser.

As expected, accuracy tends to improve as the dataset grows larger, the exception being the first extended corpus (i.e. the addition of sentences from Wikipedia) when targeting only the top-10 verb types.

In most of the cases, SVMTool performs better than SVM-TK. The gap between both approaches decreases as the datasets grow. This is natural, since accuracy becomes harder to improve on as it gets higher and, as such, equal amounts of additional data have diminishing returns.

Another contributing factor towards the gap between the approaches may be that, even with the extended datasets, there is still not enough data to allow SVM-TK to really come into its own. A similar effect was seen in previous experiments, for instance when SVMTool trailed behind TnT, up to a point where, when additional training data were used, SVMTool surpassed TnT as the best sequential tagger.

We believe that this claim is given credence by the performance comparison when targeting only the top-10 verb types. In this case, there are enough training instances of each type in the dataset to allow the ensemble of classifiers that supports SVM-TK to discriminate between the possible alternatives, and we thus see SVMTool being surpassed by SVM-TK as additional data are used for training.

When targeting the top-20 or top-30 verb types, to say nothing of the case where the classifier casts an even wider net, most of the individual OvO classifiers that form the ensemble are tasked with choosing between two classes with extremely skewed distributions, when discriminating between a common type and a rare type, or tasked with choosing between two classes with very few instances. The individual classifiers in the ensemble that have

to choose between two frequent types may perform better, but many of these classifiers are already present in the top-10 ensemble.

5.7 In-grammar disambiguation

At this juncture, several approaches have been tested as a way of improving on the standard mechanism of OOV handling, which consists of assigning a generic (or default) type that depends on the POS category of the OOV word. What all these approaches have in common is that they move the task of disambiguating among the possible lexical types out of the grammar, into a pre-processing step.

This raises a pertinent question: Perhaps all this machinery that is used to assign types in a pre-processing step is not needed after all. Maybe it can simply be discarded and the decision on which is the correct type left to the disambiguation module of the grammar, similarly to what was done in (van de Cruys, 2006) with the Alpino grammar for Dutch (cf. §2.1.3, page 21).

Following the approach of in-grammar disambiguation may seem the best choice since, in theory, it allows integrating information from all aspects accessible to the grammar into the statistical model used by the disambiguation component while, for instance, the SVM-TK classifier presented earlier is only able to look at the grammatical dependencies.

In this Section we test the performance of using in-grammar disambiguation to resolve OOV words and show that this approach is unable to cope with the vast amounts of ambiguity in the problem space.

The experimental procedure for testing the performance of the grammar when coupled with its disambiguation module is based on explicitly introducing lexical ambiguity by replacing entries in the lexicon of the grammar by multiple entries that only vary in terms of their lexical type.

An entry from the lexicon of the grammar is replaced by n entries, one for each of the top- n most frequent deep lexical types pertaining to the POS category of that word. When the grammar is run over the raw sentences in the dataset it will arrive upon an occurrence of a word corresponding to an entry that was replaced. At that point, all of the multiple entries for that word will be triggered and the disambiguation module of the grammar will eventually have to choose a deep type from among the n types that are possible for that word. To determine disambiguation accuracy, the type assigned by the grammar is compared with the correct type in the grammatical representation stored in CINTIL DeepBank.

	top-10	top-20	top-30
SVM-TK	90	82	80
in-grammar	43	20	16

Table 5.10: Accuracy (%) when assigning from the top- n verbal types

Note that, for the sake of comparability with the pre-processing approach, the experiment with the grammar only targets verbal types. Note also that the underspecified top- n lexical entries are not created for every word in the lexicon. For instance, “non-content” verbs, such as those that enter auxiliary verb constructions,¹³ are left unaltered in the lexicon since they are extremely frequent and form a well-defined closed list. It is thus reasonable to assume that such entries would always be present in the lexicon of any grammar and would never need to be treated as being OOV.

Table 5.10 summarizes the accuracy results that were obtained in this experiment. Note that the results shown for the SVM-TK classifier are lower than the ones mentioned previously in Table 5.5 (page 92). This happens because, in the previous case, top- n accuracy was given over all verbs bearing those types, while in the current case, as mentioned above, certain common verbs (e.g. auxiliary verbs) are not considered. Since the verbs that were left out have high frequency, they tend to be tagged more accurately, leading to a better overall score in the previous case.

It is clear that the approach of using the grammar to assign deep lexical types to OOV words performs quite poorly when compared with using the classifier.

This likely happens because the disambiguation module is unable to cope with the huge amount of ambiguity that is introduced when each verb is allowed to have n types. Even the smallest search space that was tested, where n is 10, is vastly superior to the amount of ambiguity present in the original lexicon, where 81% of the verbs only have a single type, and where there are no verbs with more than three types.¹⁴

The results show that relying solely on the grammar for disambiguation is not a feasible approach.

¹³Verbs like *ser/estar* (to be) and *ter/haver* (to have).

¹⁴There might also be cases of sentences with more than one unknown verb, which would have an even greater detrimental effect due to compounding ambiguity, though the rather short sentence size limits this to some extent.

	top-10	top-20	top-30
median	1.6	3.9	5.9
mean	12.3	25.6	35.3
worst case	834.1	953.9	991.6

Table 5.11: Extra memory (MB) needed for top- n verbal types

Time and memory requirements

Though it is not the main concern in this study, it is worth noting to what extent the use of multiple (top- n) types for a word affects the processing time and memory requirements of the parsing process.

The grammar, with its original lexicon, took 337 seconds to process the sentences in v3 of CINTIL DeepBank. There is a fivefold increase in processing time when using the underspecified entries for top-10 verbal types. When using the top-20 and top-30 verb types the corpus takes, respectively, 11 and 21 times longer to parse.

Note that, when in use over “real-world” text and with an unaltered lexicon, it is unlikely that LX-Gram would find OOV words in every sentence, as it is bound to happen in this test. The increase in parsing time that was reported above is, on average, the increase for each sentence with OOV words.

Memory usage is affected in a similar manner. Table 5.11 shows the median, mean and worst case increase in memory usage of each top- n test when compared with running the grammar with its original lexicon.

For instance, when using the underspecified entries for top-10 verbal types, the PET parser uses, on average, 12.3 MB more for each parse than when using its original lexicon. Note, however, that the median value is only 1.6 MB. The mean value is skewed by some highly complex sentences that require a great deal more memory when using underspecified types. For top-10, the most extreme case is a sentence that requires an extra 834.1 MB to parse.

As such, even if we were to disregard its poor disambiguation accuracy, this approach is hardly suitable for on-the-fly processing.

5.8 Experiments over another language

The machine-learning algorithms presented in the previous Sections of the current Chapter are language-independent. Thus, evaluating them over a

dataset	sentences	tokens
CINTIL DeepBank		
base (v3)	5,422	51,483
largest extended	21,217	288,875
Redwoods	44,754	584,367

Table 5.12: CINTIL DeepBank and Redwoods

language other than Portuguese is a natural path to follow.

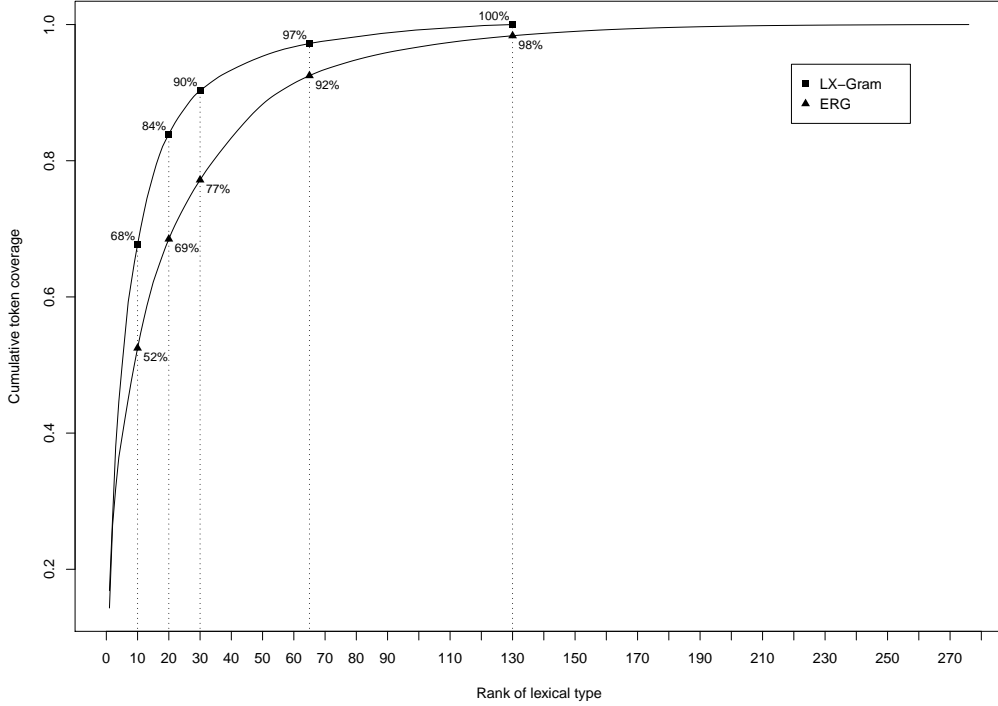
To carry out this experiment we turned to the LinGO Redwoods Treebank of English (Oepen *et al.*, 2002, 2004), a corpus composed from datasets from various domains, including e-commerce texts, the SemCor subset of the Brown corpus, articles from Wikipedia, etc. Like CINTIL DeepBank, the Redwoods corpus is a collection of manually disambiguated analyses whose construction is supported by a deep computational grammar which, in this case, is the English Resource Grammar (ERG), an HPSG for English (Flickinger, 2000).

The most recent version (Seventh Growth) of Redwoods was taken from the SVN repository where ERG is stored and, similarly to what was done previously for DeepBank, a vista with grammatical dependencies in CoNLL format was extracted for each analysis.¹⁵ The resulting corpus has a total of 44,754 sentences and 584,367 tokens, making it several times larger than the base CINTIL DependencyBank, and twice as large as the largest extended dataset used in §5.6 (cf. Table 5.12).

Despite their large difference in size, both corpora share similarities regarding the distribution of lexical types. There are 276 verb types in ERG, roughly twice as many as in LX-Gram. Their frequency in the corpus is also skewed, as one would expect, showing the usual Zipfian long tail of infrequent types (e.g. half of the verbal types account for only 1.4% of the verbal tokens).

Figure 5.5 shows a plot comparing the cumulative coverage of the top- n verbal lexical types in ERG and in LX-Gram. The topmost line shows the coverage of the types in LX-Gram, and it basically repeats what was shown in Figure 5.4 (page 90). Naturally, this line stops at 130 since this is the number of verbal lexical types in LX-Gram. Underneath that line runs the line that depicts the coverage of verb types for ERG. For any fixed value of n , the top- n types in LX-Gram provide higher coverage than the one given

¹⁵I'd like to thank Angelina Ivanova for her help in extracting the CoNLL dependencies from the Redwoods analyses.

Figure 5.5: Cumulative token coverage of top- n verbal lexical types

by the same number of top- n types in ERG.

An alternative way of comparing the coverage of verb types in both grammars is presented in Figure 5.6. The scatter plot shows for each grammar the number of types required to achieve a given coverage value. Note that the points corresponding to 0% and 100% coverage are not shown since these are trivially fixed. They are, respectively, 0 and the full set of verb types (130 for LX-Gram and 276 for ERG).¹⁶

An interesting point that is highlighted by the scatter plot is how, despite the differences in corpora and tagset size, there tends to be a stable ratio between the values of n required by ERG and LX-Gram to ensure a certain coverage. This ratio is close to the ratio between the number of verb types in ERG and in LX-Gram ($\frac{276}{130} \approx 2.12$), shown in the plot as a dashed line.

When comparing the performance of the classifier for the two grammars, setting the same top- n range for both is not suitable, since that will impose coverage ceilings that are very different. Instead, the coverage information presented previously is used to pick values n_1 and n_2 in such a way as to

¹⁶For readability, the axes in Figure 5.6 do not use the same scale. If the same scale were to be enforced, most points would be compressed in a tiny area.

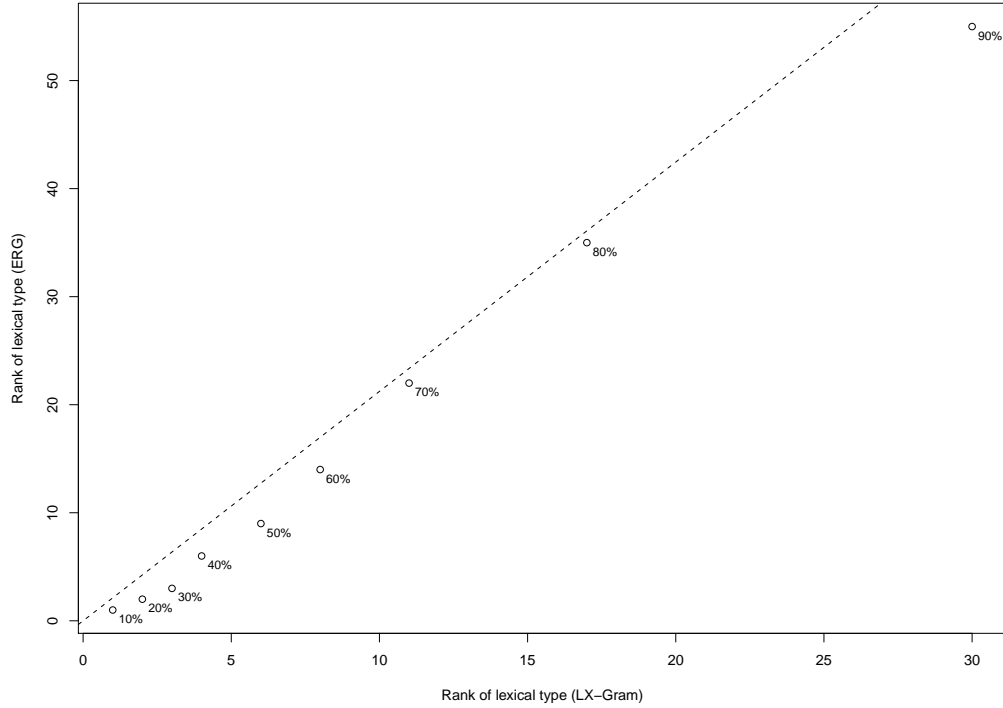


Figure 5.6: Top- n types needed for a given coverage (LX-Gram vs. ERG)

ensure that top- n_1 in LX-Gram has a coverage that is similar to that of top- n_2 in ERG.

The top- n ranges that, for ERG, have a coverage that is closest to the coverage achieved by top-10, top-20 and top-30 in LX-Gram are, respectively, top-19, top-41 and top-56.

Not all of the experiments that have been presented previously will be repeated for ERG. Instead, SVMTool is taken as a representative of the sequential supertaggers, since it achieves the best performance among those approaches.¹⁷

Table 5.13 shows the accuracy scores obtained by SVMTool (to ease comparison with the experiment for Portuguese, the LX-Gram column repeats the values from Table 5.3 and Table 5.4).

Looking at the evaluation over all tokens, or over all tokens excluding punctuation, reveals the positive impact of using the larger Redwoods corpus, which allows for higher accuracy despite the larger tagset. However, looking only at the performance over verb tokens on ERG reveals an interesting

¹⁷TnT was also tested on ERG, scoring consistently below SVMTool.

	ERG	LX-Gram
over all tokens	92.30	90.61
excl. punctuation	91.16	89.19
over all verb tokens	86.55	90.94

Table 5.13: SVMTool accuracy (%) on ERG and LX-Gram

	SVM-TK	SVMTool		SVM-TK	SVMTool
top-19	93.05	91.53	top-10	94.76	94.20
top-41	91.63	89.63	top-20	90.27	92.49
top-56	90.93	88.80	top-30	89.04	92.48

(a) over ERG

(b) over LX-Gram

Table 5.14: Classifier accuracy (%), top- n verb types, ERG and LX-Gram

point. The sharp drop in accuracy is indicative of the limitations of the model used by SVMTool. The high granularity and rich SCF information implicit in the lexical types used by ERG is not easily captured by models that rely on fixed windows of context, like the n -grams of SVMTool.

The experiment reported previously in §5.6 where the classifiers were tested over extended corpora failed to highlight this point since, while the corpora size was increased, the tagset size effectively remained the same.

Thus, testing over ERG/Redwoods is not only interesting due to the change of grammar and language, which addresses the generality of the approach, but also because it provides an account of how the performance of these classifiers changes as the grammar grows in complexity.

Table 5.14 shows the accuracy of SVM-TK and SVMTool over ERG. To ease comparison with the previous experiments, the corresponding values for performance over LX-Gram from Table 5.5 are repeated here.

Further emphasizing the results from Table 5.13, the evaluation shows that the accuracy of both approaches drops when assigning the types from ERG. The bigger dataset does not make up for the larger tagset and higher number of assignable types.

The most interesting result comes from observing how accuracy changes as the set of assignable types grows. Similarly to what happened previously with the LX-Gram experiment, the accuracy of every classifier drops as the number of top- n types being covered increases. Note, however, that on ERG/Redwoods SVM-TK consistently stays ahead of SVMTool.

From the accuracy results in Table 5.14, we know that assigning verbal types in ERG is not easier to do than in LX-Gram. The proposed explanation is that the complexity of the tagset, and that of the linguistic information that is required to build an adequate model, reached a point where it cannot be properly accounted for by the model of the sequential supertagger, requiring instead the structured information encoded in grammatical dependencies used by SVM-TK.

5.9 Summary

This Chapter presented the experiments where the classifiers that assign deep lexical types were intrinsically evaluated.

The first experiment reported in this Chapter was a preliminary exploratory trial that sought to find useful tools and feasible approaches to the task of assigning deep lexical types. Run over the stable version of CINTIL DeepBank available at the time (v2), it suffered from data-sparseness issues due to the small dataset, which had only 1,204 sentences. Nevertheless, the experiment confirmed TnT as a feasible supertagger that is able to provide a very strong accuracy baseline despite having a simple model.

An instance classifier that made use of dependency information was tested, but it performed worse than TnT, likely due to the small dataset and highly skewed distribution of lexical types. As an attempt to tackle this latter issue, the SMOTE over-sampling technique was applied, though it was not used in the subsequent experiments since the new synthetic instances it creates are not linguistically well formed.

The remaining experiments make use of the most recent stable version (v3) of CINTIL DeepBank, with 5,422 sentences. These experiments build on what was learned in the exploratory trial by reusing the most promising approaches. Some of these experiments have also been reported in (Silva and Branco, 2012a,b).

The approach that uses sequential taggers is represented by TnT, which had the best performance in the preliminary experiment, and by SVMTool, a more recent POS-tagger with competitive performance.

The SVM-TK instance classifier is presented as a way of seamlessly incorporating structured information in the features without requiring much effort in terms of feature engineering. Given that SVM is a binary classifier, an ensemble of one-vs-one classifiers was used. Since most types have only a few occurrences in the corpus, training and evaluation was performed

only for the top- n most frequent types, though different values for n were tested (viz. 10, 20 and 30). The types being assigned are also restricted to verbal types since this category tend to display richer variation in terms of subcategorization frames, these being a large part of the information encoded in a lexical type.

The SVM-TK classifier for the top-10 most frequent verb types performs better than the competing sequential supertaggers, though it falls behind when moving to the top-20 and top-30 ranges. Two possible explanations for this behavior are proposed and tested: (i) SVM-TK starts off with better results only because it is running over accurate dependency information; and (ii) the accuracy of SVM-TK drops due to data-sparseness issues.

To test the former situation, SVM-TK was trained and evaluated using features based on predicted dependencies automatically provided by MaltParser. While the noisy feature cause the accuracy of the classifier to drop, the effect is mitigated by the fact that the cases where MaltParser performs worse correspond to the less frequent types, where errors have less impact on the overall score. Also, additional training data is expected to improve not only the SVM-TK classifier, but also the underlying dependency parser.

To test the latter case, (ii), the same experiments were run over extended datasets.¹⁸ Results show that, as more training data is made available, all approaches improve their accuracy, but not in equal amounts. SVMTool pulls away from TnT, becoming the best sequential tagger. The SVM-TK classifier, running over predicted dependencies, is even better when targeting the set of top-10 verbs, though data-sparseness issues lead to it falling behind SVMTool as soon as the set of assignable verbs is increased.

The existence of a probabilistic disambiguation module in LX-Gram opens up a totally different form of handling OOV words. Instead of assigning a lexical type in a pre-processing step, a set of possible types is defined and the grammar is left to disambiguate and pick the most likely result. However, the accuracy of this approach was much worse than the one achieved by any of the pre-processing methods, in particular SVM-TK.

¹⁸The dataset was automatically extended by annotating text with LX-Gram, allowing the probabilistic disambiguation module of the grammar to pick the most likely parse for each sentence, and extracting a vista with grammatical dependencies from the result.

The final experiment consisted in repeating some of the previous experiments for a different grammar and corpus, namely the English Resource Grammar (ERG) and the Seventh Growth of the Redwoods corpus.

ERG has been under development for far longer than LX-Gram and accordingly has a richer and more granular set of lexical types, while Redwoods is much larger than the base CINTIL DeepBank corpus. Thus, this experiment allows checking how the classifiers behave when applied to a more complex tagset and being supported by additional training data.

SVM-TK was compared with SVMTool, since that latter had shown the best performance among the sequential taggers. The experiment found that, while for Portuguese SVM-TK eventually falls behind SVMTool as the set of top- n verbs is increased, in ERG/Redwoods the SVM-TK classifier is able to consistently score better than SVMTool.

The SVM-TK classifier fulfills the requirements put forward at the start of this dissertation. It is able to assign, on-the-fly, a single disambiguated lexical type, which can then be used by the grammar when faced with an OOV word. It makes use of structured information, namely grammatical dependencies, and does not require specific knowledge about the grammar or about the type hierarchy.

The SVM-TK approach is compared against using a regular POS-tagger, SVMTool, that has been trained with an enriched tagset formed by lexical types instead of POS tags.

In general, SVMTool is more accurate than SVM-TK, but the results that were obtained point towards this being a matter of data-sparseness, contingent on the corpus being used, that hampers the performance of SVM-TK, and not so much a deficiency of the approach that is followed.

The results summarized in Table 5.9 (page 96) show that, when SVM-TK assigns from the set of top-10 verb types, and is thus given enough training instances of each verb type, it performs better than SVMTool. It is only when the set of assignable types grows, without a corresponding increase in the size of the training data, and the training instances become much sparser, then SVM-TK loses to SVMTool. The results from Table 5.14 (page 103) also argue for this conclusion since they show that SVM-TK is clearly better than SVMTool on the larger (English) dataset.

Chapter 6

Parsing with Out-of-Vocabulary Words

The preceding Chapter reported on several experiments where the SVM-TK classifier was evaluated intrinsically. The main purpose of the classifier is to provide lexical types that a deep grammar can use.

This Chapter addresses the extrinsic evaluation of the SVM-TK classifier by measuring how the performance of the grammar is impacted when LX-Gram relies on SVM-TK to pick the lexical types of OOV words instead of choosing a default type (baseline).

6.1 LX-Gram with SVM-TK

As mentioned previously, LX-Gram is run over a corpus that has been annotated by a POS tagger. While the grammar ignores the POS tag for words that are in its lexicon, this setup provides the grammar with some degree of robustness since, when faced with an OOV word, the grammar falls back on the POS information, using the assigned POS tag to trigger a default deep lexical type. This is a sub-optimal solution, since for a given POS tag the grammar will always use the same lexical type, but it is better than an outright parse failure.

As an alternative, we experiment here with SVM-TK as a bridge between a richer annotation, namely grammatical dependencies, and the grammar.

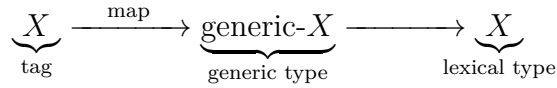


Figure 6.1: Mapping from a tag to a generic type

We wish to use the types assigned by SVM-TK, but the version of the parser we are using for this study has no straightforward mechanism to force words to receive a lexical type that is given by a pre-processing step external to the grammar. To overcome this limitation of the current system we use the POS-mapping mechanism that is already in place, known as generic types, to link the SVM-TK pre-processing step and LX-Gram.

Each tag that can be assigned by the classifier is in a one-to-one mapping with a generic type that, in turn, corresponds to a lexical type, as exemplified in Figure 6.1. Note that this workaround does not introduce anything new. From the point of view of the grammar, the tags assigned by SVM-TK are treated as POS tags that map to some lexical type, as usual. There are, however, some issues regarding efficiency, in terms of speed and memory usage. In particular, given the way the parser is implemented, using the mechanism of generic types circumvents the process of morphological analysis in the grammar. As such, the generic type must also include information about morphology. For Portuguese verbs, this means that the generic type must explicitly refer the time, aspect, mood, person and number features.¹ Due to all the possible combinations of these features, and just to account for the top-10 most frequent verb types, we need to create slightly over one thousand POS-mappings.

Future versions of LX-Gram will be supported by a more recent parser, which will allow using a different method, chart mapping, that promises to be a more elegant and efficient way of setting the lexical types of OOV words.

The extrinsic test is performed over a set of 5,000 sentences taken from articles from the Público newspaper. These sentences have not been used before, so they are “previously unseen” to every tool involved in the experiment, namely the POS-tagger, the dependency parser, the SVM-TK classifier and the disambiguation module of LX-Gram. They are, however, in the same genre as much of the corpus that was used to train these tools.

In the preceding Chapter several versions of the SVM-TK classifier were

¹These were omitted from the example in Figure 6.1 for the sake of clarity.

tested. For this experiment, given that we start with an unannotated test corpus, we must run SVM-TK over predicted dependencies.² This would nonetheless be the proper choice since the extrinsic experiment seeks to assess performance under a real usage scenario where it is unrealistic to expect that SVM-TK will receive fully correct dependency information to build the features it needs. Concerning the set of lexical types assignable by the classifier, we opted for the smallest that was tested, i.e. top-10 verb types. In the intrinsic evaluation, this classifier obtained 93.88% accuracy (cf. Table 5.6 on §5.5, page 93).

6.1.1 Coverage results

To begin the extrinsic evaluation of SVM-TK, we compare the baseline of LX-Gram running over POS-tagged text against LX-Gram running over text that has been annotated with lexical types by SVM-TK. When running in the baseline mode, OOV words will be assigned a default type. Alternatively, when coupled with SVM-TK, the grammar will use for OOV words the types assigned to them by SVM-TK. Mind that it is to be expected that, in some cases, SVM-TK will assign the same type as the default that is given in the baseline mode.

As such, each sentence can fall into one of the following 4 cases:

- case $[-]$, sentences that fail to parse in the baseline mode and also when LX-Gram is coupled with SVM-TK;
- case $[-+]$, sentences that fail to parse in the baseline mode but that receive at least a parse when LX-Gram is coupled with SVM-TK;
- case $[+-]$, sentences that receive at least a parse in the baseline mode but that fail to parse when LX-Gram is coupled with SVM-TK;
- and case $[++]$, sentences that receive at least a parse in the baseline mode and also when LX-Gram is coupled with SVM-TK.

Note that being able to assign at least a parse to a sentence does not mean that that parse, or any of the other parses in the parse forest returned by the grammar, is necessarily correct. That is, the comparison outlined above speaks only of the relative coverage of both approaches.

The number of sentences that fall into each of the 4 cases outlined above is summarized in Table 6.1.

²For this we use the dependency parser described in §5.5, trained over the totality of CINTIL DependencyBank (5,422 sentences).

case	sent.
[--]	3474
[-+]	10
[+-]	37
[++]	1479
total	5000

Table 6.1: Coverage (using SVM-TK for top-10 verbs)

In the baseline mode, 3,484 sentences, or roughly two-thirds of the total sentences, do not receive any parse. This is in line with what was expected, given LX-Gram has a coverage of around 30%. When using SVM-TK, 10 of these sentences receive at least a parse (case [-+]). Conversely, of the 1,516 sentences that receive at least a parse in the baseline mode, 37 drop from the coverage of the grammar when coupled with SVM-TK (case [+−]), leading to an overall loss in coverage of 27 sentences.

6.1.2 Correctness results

Coverage scores alone paint an incomplete picture of the extrinsic evaluation task that is intended.

In case [-+], there are 10 sentences that receive no analysis in the baseline mode but that receive analyses when using the types assigned by SVM-TK. However, we must determine how many of the 10 resulting parse forests contain the correct parse. Conversely, [+−] suggests that 37 sentences were lost. However, we must determine in how many of those situations the correct parse is actually present in the parse forest in the baseline mode.

Finally, though in case [++] there is no change in coverage, we still need to take a closer look at the parse forests since there are two distinct situations that may occur: Either (i) SVM-TK has assigned the same type as the default that was given in the baseline mode, in which case there is no difference between the parse forests; or (ii) SVM-TK has assigned a type different from the default one, in which case we need to determine which parse forest, if any, contains the correct parse.

Accordingly, the extrinsic evaluation task requires manual analysis of the parse forests, following the grammar-supported treebanking approach described in §4.2, to determine whether the correct parse is present or not.³

³I'd like to thank the annotators, Helena Pimentel and Luís Morgado, and the adjudicator, Francisco Costa, for their help.

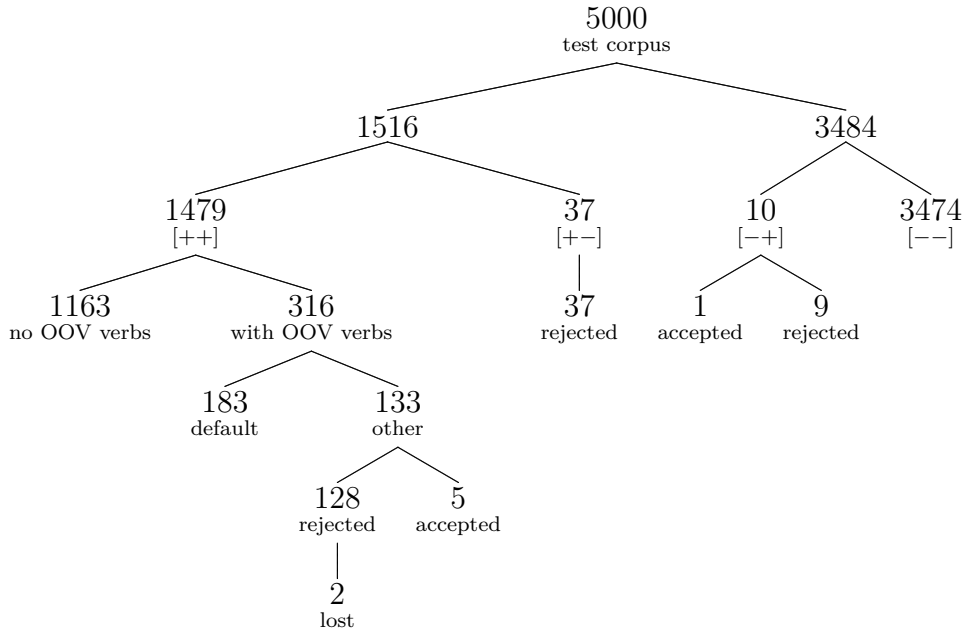


Figure 6.2: Breakdown of sentences in the extrinsic experiment

Note that, for the sake of efficiency, the parse forests produced by LX-Gram are limited at the first 250 parses, according to the ranking given by the disambiguation module of the grammar.

The following discussion of the extrinsic evaluation results will be split along the cases described above. Figure 6.2 gives an overview.

For 3,474 sentences, or about two-thirds of the corpus, the grammar fails to deliver any parse, both when running in the baseline mode and when running over types provided by SVM-TK. Since the grammar does not produce any output for these cases, no manual treebanking can be done.

Case $[-+]$ concerns those sentences that receive no analysis when in the baseline mode and that become parsable when the grammar uses the types assigned by SVM-TK. There are 10 sentences that fall into this case. For 9 of them the annotators reject every parse in the parse forests, which leaves a single sentence where the increase in coverage results in a new parse being accepted.

Case $[+-]$ is the converse of the previous case. It concerns those sentences that are covered by LX-Gram in the baseline mode but that do not receive any analysis when using the types assigned by SVM-TK. Despite the drop of 37 sentences in absolute coverage, none of the parse forests produced by LX-Gram in the baseline mode contain analyses that are accepted by the

annotators, so no correct parses are lost. This gives a net benefit in terms of correctness.

In the final case, $[++]$, we find 1,479 sentences that are parsable in the baseline mode and also when using the types assigned by SVM-TK. A sizable chunk of 1,163 sentences can bypass manual evaluation since its sentences do not involve any verbal default types (i.e. they have no OOV verbs), and therefore the correctness of the resulting analyses will not be affected by using the types assigned by SVM-TK. This leaves 316 sentences that can potentially be affected by the use of SVM-TK.

This set of 316 sentences is further reduced by leaving aside the 183 sentences where SVM-TK assigns to OOV words the same type as the default type since in those cases the correctness of the resulting analyses does not change. In this way, the number of sentences in $[++]$ that need to be manually analyzed is reduced to 133.

To restate, these 133 sentences are those that receive at least a parse in the baseline mode and also when using the types assigned by SVM-TK, but where the type assigned by SVM-TK is not the default type used in the baseline mode.

Through the treebanking process we find that, of those 133, nearly all (128) are rejected and, again, nearly all (126) are cases that are also rejected in the baseline mode, so the acceptability of these analyses does not change when using the type assigned by SVM-TK instead of the default type. This might happen because the default type and the type assigned by SVM-TK are both wrong; or because, despite one of those two types actually being correct, there being some other reason not related to OOV verbs that prevents accepting the parse.

Out of the 133 sentences, this leaves 2 with accepted parses in the baseline mode that, due to SVM-TK assigning a type other than the default type, end up being rejected. Conversely, there are 5 sentences whose parses are rejected in the baseline mode that have accepted parses when using the types assigned by SVM-TK.

6.1.3 Discussion

Using the types assigned by SVM-TK instead of relying on the default type as a way of dealing with OOV words leads to minor improvements in the performance of the grammar. Note that here we refer to performance in terms of coverage and correctness. Performance in terms of parsing speed and memory usage actually becomes worse when using SVM-TK since the mappings from tag to type increase greatly in number and the algorithm that does that mapping seems to be quite inefficient.

The coverage and correctness improvements are minor but are found in several places.

Case $[-+]$ concerns what may be seen as the standard OOV situation where LX-Gram fails to assign a parse in the baseline mode but succeeds when using SVM-TK. Out of the 10 sentences that gain analyses, only 1 has an accepted analysis. We find that, in most of the other 9 cases, SVM-TK assigns a type intended for copula verbs⁴ to verbs that cannot ever be a copula. This issue will be addressed further ahead.

In case $[+-]$, we find 37 sentences that drop from the coverage of the grammar when we switch to using the types assigned by SVM-TK. The subsequent analysis through manual treebanking shows that none of those 37 sentences has accepted parses to be lost. In this regard, SVM-TK acts as a filter, easing the work of the human annotators by reducing the amount of parse forests that they must analyze.

Case $[++]$ concerns another aspect of OOV words, where the missing word does not cause a parse failure but SVM-TK may present a better alternative than the default type. Though there are only a few cases, we find that using the type assigned by SVM-TK is the better solution.

The following paragraphs address several factors that we envision may have impeded SVM-TK from having a stronger impact on the extrinsic performance of the grammar.

We begin by recalling the fact that much of the parse ambiguity does not involve the lexicon at all. As pointed out in §2.2.2, a study performed on ERG by Toutanova *et al.* (2002) found that, even when using an oracle tagger that perfectly assigns a fully disambiguated deep lexical type to every word, the disambiguation module in ERG ranks the correct analysis in first place only for 55% of the sentences.⁵ This issue is not specific to SVM-TK, and places a hard ceiling on the precision that can be achieved.

In the experiment reported here, manual treebanking was used to assess only whether the correct parse was present in the parse forest, not whether it was ranked first. As such, parse ranking only has a negative impact when the correct parse is ranked so low as to be outside the 250 most likely analyses that form the parse forest. In other scenarios, e.g. an application that only takes the top-ranked analysis, failing to rank the correct parse in the first position would have a greater negative impact.

Another issue, also not specific to SVM-TK, relates to the completeness

⁴The verbs *ser* (Eng.: to be, as an essential characteristic) and *estar* (Eng.: to be, as a state or situation).

⁵We ran a similar test for LX-Gram and found a value of 56%.

of the grammar and of the type hierarchy. When faced with an OOV word, the grammar may fail to produce the correct parse due to a number of reasons. In particular, it may happen that the correct type for that word in that context does not even exist in the type hierarchy. Note that this is not the case of the correct type being so rare as to be outside the set of top- n most frequent types. Instead, there is no deep lexical type in the type hierarchy that can be used for that word in that context. As such, until the type hierarchy is extended with the proper type, sentences that include that word in that context will not be analyzed or, if an analysis is generated, it will always be rejected during manual treebanking.

The distribution of lexical types is highly skewed. Accordingly, it is expected that the SVM-TK classifier will assign the most frequent type more often than the other types. This type corresponds to the default type that LX-Gram uses in the baseline mode for OOV words. We thus see an appreciable amount of cases (183 out of 316, cf. Figure 6.2) where relying on SVM-TK is no different from using the baseline setup.

Finally, we note that SVM-TK may be trying to assign types that are not relevant for OOV words. One of the guiding principles of the approach that is proposed in this work is that the classifier should try to be as agnostic as possible in what concerns the grammar and its type hierarchy. Taking the top- n most frequent types without regard to what they stand for is a consequence of this decision. While this means that this approach can be applied to any grammar without having to know the particular details of how that grammar is implemented and how the type hierarchy is organized, it has a downside in that it might try to assign types that are unlikely to be useful for OOV resolving.

This may be the case with deep lexical types with a very small lexical diversity. That is, types that cover only very few words in the lexicon and, more to the point, it is expected that the lexicon already contains all possible words that bear those types.

Taking LX-Gram and SVM-TK as an example, a look at the set of the top-10 most frequent verb types will reveal the presence of types used solely for copula verbs. Given their importance, it is to be expected that all such verbs will already be present in the lexicon of LX-Gram and thus that there will never occur an OOV verb that should be assigned a copula type. However, due to the high frequency of these types in the training data, SVM-TK will often assign them to other, non-copula verbs. Also, since we are working with a limited set of top- n types, having these types be part of the category space assignable by the classifier means that other types will necessarily be left out.

Note, however, that this effect will tend to become less pronounced as

the set of assignable types grows (i.e. as the n in top- n increases). Since the number of types with very limited lexical diversity is rather small, they will make up a smaller portion of the assignable type space when larger top- n sets are used.

6.2 Summary

In this Chapter we presented results from the extrinsic evaluation of the SVM-TK classifier. LX-Gram was run in the baseline mode, over POS-tagged text, and also run using SVM-TK (top-10 verb) as a classifier OOV types, and the evaluation looked for changes in coverage and parse correctness.

Results show a minor impact on the coverage of the grammar and on the correctness of the parses that are generated. On a test set of 5,000 sentences, 3,484 are not parsed by LX-Gram in the baseline mode, a value in line with the about 30% coverage the grammar is known to currently achieve. When using the types assigned by SVM-TK, 10 more sentences are covered, of which only 1 has a parse that is accepted by the annotators. Conversely, using SVM-TK leads to 37 sentence dropping out from the coverage of the grammar, though treebanking reveals that no acceptable parses were lost in those 37 sentences.

There are 1,516 sentences that are covered by the grammar, whether in the baseline mode or with SVM-TK. For these sentences, 2 have their parses become rejected when using the classifier, while there are 5 where the opposite happens, i.e. sentences whose parses are rejected in the baseline mode that become acceptable when using SVM-TK.

Chapter 7

Conclusion

This Chapter concludes the dissertation. It begins with a Section where the most relevant points from the preceding Chapters are summarized. This is followed by some concluding remarks that put those results in perspective and by comments on future work.

7.1 Summary

Deep computational grammars are highly prized due to the precise and detailed way in which they account for highly complex linguistic phenomena. They are used whenever a precise grammatical analysis is required.

Such approaches are not fully robust in the face of malformed input or, in the particular case of highly lexicalized grammar frameworks like HPSG, when dealing with words missing from their lexicon, i.e. out-of-vocabulary (OOV) words.

The lexicon of such grammars is highly complex (cf. Appendix B) and, as it is well known, no matter how large the lexicon is, there will always be OOV words due to the novelty and lexical creativity that is intrinsic to natural languages.

Therefore, regarding the handling of OOV words, these deep grammars can greatly benefit from an integration with robust approaches which can be used to annotate the word that is unknown to the grammar with some sort of linguistic information, such as part-of-speech (POS) category, sub-

7. CONCLUSION

categorization frame (SCF) restrictions, and more, that allow the grammar to take the word as if it had been present in its lexicon all along (with the important caveat that the annotation process may not be fully accurate).

Probabilistic POS taggers have often been used to provide an initial annotation upon which a parser can build its analysis. For instance, LX-Gram, the HPSG working grammar used in our research, currently relies on the annotation provided by a plain POS tagger as a way of handling OOV words. Such a coarse category allows LX-Gram to trigger a default lexical type for that category that, even though it might be the best choice (i.e. the most likely type) for that category, will nevertheless be wrong in many cases. Consequently, having a deep computational grammar as the ultimate consumer of the annotation that is produced brings about the specific challenge of being able to generate annotation that is rich and detailed enough to be useful for the parsing process supported by the grammar.

To handle OOV words, the initial annotation provided by the probabilistic process needs to have increased granularity as to match the range of possible deep lexical types. Some of the finer grammatical details that allow to assign a deep lexical type are hard to capture with models that only rely on features taken from a fixed window of context, as commonly found on current approaches.

Since grammatical dependencies occurring among word tokens in a sentence are an instantiation of the SCF of a target OOV word, they should provide information that is useful to discriminate between deep lexical types. This is the rationale that motivated the creation of a classifier that is able to use information about grammatical dependencies.

Note that the decision on the type of linguistic information to use is contingent on available resources. Naturally, we would not be able to use grammatical dependencies if a corpus annotated with that information and a trainable dependency parser were not available. However, there are several machine-learning parsers with state-of-the-art performance that are freely available and the corpus used in these experiments contains information on grammatical dependencies.

Tree kernels were chosen as a principled way of seamlessly integrating an encoding of grammatical dependencies into a support-vector machine classifier, SVM-TK. This classifier was compared with an approach where a POS tagger is repurposed to assign deep lexical types, since that task can be seen as doing POS tagging with a highly granular tagset. The model underlying SVM-TK is a support-vector machine, a binary classifier. As such, SVM-TK is designed as an ensemble of several one-vs-one classifiers, one for each possible pairing of the lexical types being assigned.

The corpus used in these experiments, DeepBank, was built through manual disambiguation of the parse forest produced by LX-Gram when run over a set of (mostly newspaper) texts. Since the representation thus produced is specific to the HPSG framework and unwieldy to work with, a novel *vista extraction* mechanism was developed.

Vista extraction allows taking the full grammatical representation that is produced by the grammar and separate only the information that is relevant for the task at hand. In this case, this consists in extracting a vista with grammatical dependencies.

As expected, the distribution of types in the corpus is highly skewed. This fact, together with the modest size of the dataset, means that for many types a classifier cannot be induced since there are too few occurrences of those types in the corpus. Accordingly, SVM-TK is experimented with for a subset of types that contains only the most frequent (top- n) types in the corpus (cf. Appendix A).

Evaluation results show that SVM-TK takes the lead when running over the top-10 most frequent verb types, achieving an accuracy of 94.76%. It falls behind the other approaches (i.e. the repurposed POS taggers) when the set of types being assigned grows to include the top-20 and top-30 most frequent verb types. Given these results, two additional experiments were pursued.

A first experiment sought to ascertain to what extent the accuracy score would be affected by running the classifier using features based on grammatical dependencies assigned automatically by a dependency parser, thus approximating a realistic usage scenario. The decrease in accuracy was between 0.88 and 1.05 percentage points.

A second experiment sought to determine the impact of additional training data. This extended training data was obtained by running LX-Gram over additional text and letting the disambiguation module in the grammar pick the best analysis. The increased amount of training data allows SVM-TK to perform better than the other approaches, even when running over predicted dependencies, with verbs from the set of top-10 verb types, reaching 94.55%, 0.48 percentage points above SVMTool, when using the largest dataset. When the number of types being assigned grows, the SVMTool POS tagger performs better. Over the top-30 verb types, using the largest dataset, SVM-TK scores 91.50%, 2.51 percentage points behind SVMTool.

The disambiguation module in LX-Gram, which was used to pick the best analysis when creating the extended datasets, can also be used to tackle OOV words by allowing an OOV word to have a set of possible

7. CONCLUSION

types (i.e. taking it as a n -way ambiguous word) and accepting the reading picked by the disambiguation module. This approach requires no tools other than the grammar itself (as long as it includes a disambiguation module). However, it shows much worse performance than SVM-TK.

SVM-TK was tested with the ERG grammar for English, using the Redwoods corpus as the dataset. This allowed evaluating the classifier over a different language, and also over a larger tagset and a larger corpus.¹

Comparing against SVMTool, which shows the best accuracy among the sequential taggers, it was found that SVM-TK was able to perform consistently better, achieving 90.93% accuracy when assigning from largest (top-56) set of verb types, 2.13 percentage points above SVMTool.

The final experiment consisted of an extrinsic evaluation. For this, LX-Gram was used to annotate a new corpus of 5,000 sentences. The baseline mode, running over POS-tagged text and relying on a generic (or default) type, is compared with using SVM-TK as a pre-processor for assigning deep lexical types.

Note that there are two issues that impose a performance ceiling on the extrinsic experiment. First, we know that assigning correct lexical types alone is not enough to ensure that the correct parse will be selected or even produced. Second, the grammar and type hierarchy are not complete. As such, in some cases, there is no existing deep lexical type that can be assigned that will lead to an analysis being accepted.

A process of error analysis is needed to probe the roughly 70% of sentences that lie outside the coverage of the grammar.² This process is outside the scope of the current work, falling under the umbrella of grammar development, since it helps direct the development of the grammar and lexicon towards the missing features. Nonetheless, it can shed some light on the causes of those parsing failures and allow us to determine how many of those failures are due to OOV words. This, in turn, would allow us to better calibrate our expectations regarding what OOV handling can do.

Coverage was measured to determine if using SVM-TK changed the number of sentences that receive parses. In absolute terms, coverage went down (10 parses gained, 37 parses lost). Manual treebanking was used for a more fine-grained evaluation, and found that none of the 37 parses that dropped out of coverage were correct, giving a net benefit in terms of correctness. Out of the 10 new parses, 1 was, in the end, accepted.

¹ERG has roughly twice as many verb types as LX-Gram. Redwood is roughly twice as large as the largest extended dataset used for Portuguese.

²The error analysis process can be semi-automated by using error mining techniques, like the one described in (van Noord, 2004), which automatically finds strings that tend to occur in the unparsable sentences.

SVM-TK also has an impact on those sentences that always stay in coverage when it assigns a type other than the default one. For these cases, 2 accepted parses were lost and 5 new parses were accepted.

7.2 Concluding remarks and future work

Results have shown that, given a set of n allowable types for an OOV word, it pays off to perform the disambiguation outside of the grammar, as a pre-processing step, rather than letting that n -way ambiguous word be resolved by the grammar alone.

The main hypothesis guiding this work is that current methods for performing this sort of OOV handling in a deep computational grammar can be improved by including features derived from linguistically informed structure, in particular those derived from grammatical dependencies, since dependencies closely mirror the SCF of a word, which constitutes a large part of the linguistic information encoded into a deep lexical type.

We developed a novel approach where a classifier, SVM-TK, through the use of tree kernels, is able to use grammatical dependencies as features and provide the grammar with fully disambiguated deep lexical types that the grammar can use when faced with an OOV word.

This is a data-driven approach, independent of the language and of the grammar. It requires that the lexical information needed by the grammar be represented by a tag. This is a natural fit to the concept of lexical types in HPSG, but is also applicable to other grammatical frameworks. Given that SVM-TK generates its features from grammatical dependencies, when assigning types it takes as input text that has been automatically annotated by a dependency parser. The approach itself, however, is not limited to only making use of this type of linguistic information.

In this sense, SVM-TK acts as a bridge between processes that annotate the text, and the grammar. A bridge that takes as input the annotation produced in those shallow processes, which in the current case is a graph of grammatical dependencies, and produces as output a deep lexical type.

In some cases SVM-TK has lower accuracy than a classifier that uses less grammatically sophisticated features, here represented by the SVMTool POS tagger. However, the analysis of the results points towards data-sparseness as the main cause for the lower performance of SVM-TK in these cases since, when targeting types with a less sparse distribution, or when using a larger training dataset, SVM-TK achieves the best accuracy scores among the approaches that are tested.

The existence of a data-sparseness issue does not invalidate the approach

that is proposed here. The distribution of lexical types will always be highly skewed, so it is expected that the rarer types will typically have too few training instances to be effectively learned. Much of the data-sparseness issue is contingent on the size of the corpus being used and can be progressively mitigated by the use of additional data.

Future work

In keeping with the principle that the classifier should be as agnostic as possible regarding the grammar and the type hierarchy, SVM-TK targets the set of the n most frequent verbal types (top- n). As seen in the previous Chapter, this may happen to be sub-optimal since there are types in that set that are unlikely to ever be applicable to OOV words. A solution is, naturally, to consult a linguist knowledgeable in the grammar and ask them which types should be left out of OOV handling, but there is also the possibility of devising a way of automatically finding such types through a metric of lexical diversity, and working only with those considered to be sufficiently diverse to warrant being in the set of assignable types.

SVM-TK assigns fully disambiguated deep lexical types, freeing the grammar from having to deal with lexically ambiguous OOV words. It would be interesting to study whether there are cases where allowing some degree of ambiguity to pass through to the grammar would be advantageous. For instance, in cases where the various classifiers in the SVM-TK ensemble have little confidence on their choice, or when the voting procedure does not pick a clear, stand out winner.

This dissertation targets only OOV verbs. Now that the classification harness is in place and working, a natural path to follow is to extend OOV handling to the other open classes, namely nouns and adjectives, in order to cover the full range of categories of OOV words.

The approach proposed here consists of a classifier that lies between the processes that run prior to the grammar, whose output it uses as features, and the grammar, which, when faced with an OOV word, relies on the tag assigned by the classifier. Crucially, the method used in this approach provides a principled way of exploring how different kinds of linguistic information impact on the accuracy of OOV handling by experimenting with the features used by the classifier.

The SVM-TK classifier described in this work uses only features based on grammatical dependencies, since these were a prime candidate for improving

the ability to discriminate between verbal lexical types. In this respect alone there is already much that can be experimented with, like testing dependency features that do not include the word, omitting the word only for the open class dependents, etc.

A deep lexical type encodes much more than the SCF of a word. In this respect, a look at Appendix B is instructive in that it gives an idea of the richness and extent of the linguistic information in a lexical type. Since different representations bring to the forefront different linguistic phenomena, it would be interesting to study how grammatical dependencies can be complemented by other kinds of linguistic information.

For instance, the abstraction provided by grammatical dependencies draws attention to the relations between words that may be far apart, spanning intervening constituents. The very same abstraction, however, hides from view the structural differences between the active and passive forms of a sentence. As such, experimenting with bringing into SVM-TK features based on syntactic constituency, where these differences are explicit, may improve the classifier.

Likewise, finding ways to bring into the classifier a source of information to address the semantic properties encoded in a lexical type should prove valuable. For this, SVM-TK could include features based on a measure of similarity between words over, say, a WordNet-like ontology or some other resource containing information on lexical semantics.

The large number of different linguistic phenomena that need to be accounted for in a deep lexicon may suggest a different approach towards building the classifier. Instead of having an ensemble of classifiers trained to discriminate between deep lexical types, we create an ensemble of classifiers where each classifier is focused on a particular linguistic phenomena. Using Appendix B as a guide, this would mean having a classifier for deciding whether a verb is passivizable, another for assigning one of the 14 possible values for alternation, etc.

After all the phenomena classifiers have assigned a value, the result is mapped into a deep lexical types, just like it happens with the entries in the lexicon.

This approach has a few drawbacks: (i) it is possible that the ensemble of classifiers will arrive at a combination of values that is not consistent and does not correspond to any existing type, while with SVM-TK the type that is assigned will always correspond to a type known to the grammar; (ii) by doing this, the classifier loses some of its agnosticism regarding the grammar, since we must know what are the different phenomena and their possible values; and (iii) instead of a straightforward ensemble of homogeneous

7. CONCLUSION

classifiers, this approach relies on an ensemble of specialized classifiers, each possibly very different from the others in terms of the algorithm and features that are used, a setup which is likely to be more demanding to build.

Nonetheless, we can envision some possible important advantages to having classifiers that focus on the different phenomena.

With SVM-TK, the number of classifiers in the ensemble varies with the set of top- n verbs being considered, while with this alternative approach the number of classifiers in the ensemble remains fixed, since the number of phenomena being accounted for does not change. Accordingly, this alternative should prove to be more scalable.

Since each classifier in the ensemble specializes in a particular linguistic property, it can use only the features deemed relevant for discriminating between the possible values of that property. For instance, taking the suggestion mentioned above, the classifier for SCFs would take grammatical dependencies, while the classifier that determines whether the verb is passive could disregard dependencies and look only at syntactic constituency.

Also, many different deep lexical types have commonalities (e.g. all are passivizable or all are acceptable in absolute participles). In SVM-TK, those types have to be learned separately, while in this alternative approach the instances that belong to those different types will be taken together when training the classifiers.

This suggests that the alternative approach could also be more resistant to data-sparseness issues.

Appendix A

Verbal deep lexical types

Below we list the 130 verb deep lexical types that occur in the corpus (v3 of CINTIL DeepBank), ranked from the most to the least frequent.

- 1 verb-dir_trans-lex
- 2 verb-individual_lvl_copula-lex
- 3 verb-identity_copula-lex
- 4 verb-ditrans-lex
- 5 verb-subj_raising-comp_vp-lex
- 6 verb-dir_trans-indef_null_obj-lex
- 7 verb-stage_lvl_copula-lex
- 8 verb-intrans-obl_direction-lex
- 9 verb-dir_trans-expletive_subj-lex
- 10 verb-intrans-comp_pp_de-lex
- 11 verb-dir_trans-subj_np_inf_cp+conj-lex
- 12 verb-comp_cp-comp_indir-lex
- 13 verb-subj_control-lex
- 14 copular-verb-predicational-lex
- 15 verb-unaccusative-lex
- 16 verb-anticausative-lex
- 17 verb-intrans-lex
- 18 verb-trans-comp_pp_de-lex
- 19 verb-comp_np_inf_cp+ind_declarative-lex
- 20 verb-compound_tense_aux-lex

- 21 verb-subj_raising-comp_pp_de-lex
- 22 verb-comp_cp-lex
- 23 verb-trans-comp_pp_em-lex
- 24 verb-anticausative-optional_inherent_clitic-lex
- 25 verb-intrans-obl_location-lex
- 26 verb-intrans-comp_pp_acercade_de_em_sobre-lex
- 27 verb-intrans-comp_pp_em-lex
- 28 verb-subj_raising-comp_pp_a-lex
- 29 verb-trans-obl_location-lex
- 30 verb-intrans-obl_direction_em-lex
- 31 verb-comp_np_inf_cp+ind_declarative-comp_indir-lex
- 32 verb-ind_trans-lex
- 33 verb-anticausative-inherent_clitic-lex
- 34 verb-intrans-comp_pp_a-lex
- 35 verb-comp_np_inf_cp+conj_declarative-lex
- 36 verb-anticausative-subj_np_inf_cp+conj-inherent_clitic-lex
- 37 verb-unaccusative-ind_obj-lex
- 38 verb-trans-comp_pp_para-lex
- 39 verb-intrans-comp_pp_com-lex
- 40 verb-trans-comp_pp_por-lex
- 41 verb-subj_raising-comp_pp_por-lex
- 42 verb-dir_trans_or_ind_trans-lex
- 43 verb-trans-obl_direction_em-lex
- 44 verb-inherent_clitic-comp_pp_de-lex
- 45 verb-comp_np_or_pp_com-lex
- 46 verb-inherent_clitic-expletive_subj-comp_pp_de-lex
- 47 verb-inherent_clitic-comp_pp_a_de-lex
- 48 verb-subj_control-subj_np_inf_cp+conj-comp_vp_np_cp+ind-lex
- 49 verb-intrans-comp_pp_a_contra-lex
- 50 verb-anticausative-inherent_clitic-comp_pp_em-lex
- 51 verb-trans-comp_pp_a-lex
- 52 verb-0place-lex
- 53 verb-inherent_clitic-comp_pp_a-lex
- 54 verb-anticausative-subj_np_inf_cp+conj-optional_inherent_clitic-lex
- 55 verb-trans-comp_pp_a_em-lex
- 56 verb-intrans-comp_pp_de_em-lex
- 57 verb-inherent_clitic-comp_pp_em-lex
- 58 verb-trans-comp_pp_em_por-lex
- 59 verb-trans-comp_pp_com-lex
- 60 verb-intrans-comp_pp_com_contra_em_para-lex
- 61 verb-inherent_clitic-lex

62 verb-comp_np_inf_cp_declarative-lex
63 verb-subj_control-2comp_np+pp_a-lex
64 verb-intrans-comp_pp_em_sobre-lex
65 verb-inherent_clitic-comp_pp_em_entre-lex
66 verb-inherent_clitic-comp_pp_com-lex
67 verb-obj_raising-comp_pp_de-lex
68 verb-intrans-comp_pp_a_com-lex
69 verb-inherent_clitic-comp_pp_contra-lex
70 verb-trans-comp_pp_sobre-lex
71 verb-trans-comp_pp_com_por-lex
72 verb-trans-comp_pp_com_de-lex
73 verb-subj_np_inf_cp+conj-trans-comp_pp_em-lex
74 verb-subj_np_inf_cp+conj-trans-comp_pp_a+dat_de-lex
75 verb-subj_np_inf_cp+conj-comp_np_inf_cp_declarative-lex
76 verb-subj_control-no_cp_comp-lex
77 verb-intrans-subj_np_inf_cp+conj-lex
78 verb-intrans-comp_pp_com_por-lex
79 verb-intrans-comp_pp_com_de-lex
80 verb-intrans-comp_pp_a_em-lex
81 verb-intrans-comp_pp_a_de-lex
82 verb-intrans-comp_pp_a_de_em-lex
83 verb-intrans-comp_pp_a_contra_para-lex
84 verb-inherent_clitic-comp_pp_por-lex
85 verb-inherent_clitic-comp_pp_afavorde_contra_sobre-lex
86 verb-comp_cp_interrogative-comp_indir-lex
87 verb-anticausative-obl_location-lex
88 verb-anticausative-inherent_clitic-obl_location-lex
89 verb-trans-obl_direction-lex
90 verb-trans-comp_pp_de_em-lex
91 verb-trans-comp_pp_com_em_por-lex
92 verb-trans-comp_pp_a_com-lex
93 verb-intrans-comp_pp_para-lex
94 verb-intrans-comp_pp_em_para_por-lex
95 verb-intrans-comp_pp_contra_por-lex
96 verb-intrans-comp_pp_com_contra-lex
97 verb-intrans-comp_pp_a_de_entre-lex
98 verb-inherent_clitic-comp_pp_em_entre_para-lex
99 verb-inherent_clitic-comp_pp_a_em_sobre-lex
100 verb-trans-comp_pp_a+dat_de-lex
101 verb-obj_control-comp_pp_a-lex
102 verb-intrans-comp_pp_sobre-lex

A. VERBAL DEEP LEXICAL TYPES

- 103 verb-intrans-comp_pp_por_sobre-lex
- 104 verb-intrans-comp_pp_por-lex
- 105 verb-intrans-comp_pp_entre-lex
- 106 verb-intrans-comp_pp_em_entre_por-lex
- 107 verb-intrans-comp_pp_em_entre-lex
- 108 verb-intrans-comp_pp_de_por-lex
- 109 verb-intrans-comp_pp_a_sobre-lex
- 110 verb-intrans-comp_pp_afavorde_contra-lex
- 111 verb-intrans-comp_pp_acercade_de_sobre-lex
- 112 verb-inherent_clitic-comp_pp_para-lex
- 113 verb-inherent_clitic-comp_pp_de_por-lex
- 114 verb-inherent_clitic-comp_pp_de_em-lex
- 115 verb-inherent_clitic-comp_pp_contra_em-lex
- 116 verb-inherent_clitic-comp_pp_com_por-lex
- 117 verb-inherent_clitic-comp_pp_com_em-lex
- 118 verb-inherent_clitic-comp_pp_a_para-lex
- 119 verb-comp_np_or_pp_sobre-lex
- 120 verb-comp_np_inf_cp+conj_declarative-comp_indir-lex
- 121 verb-anticausative-subj_np_inf_cp+conj-lex
- 122 verb-trans-comp_pp_de-obligatory_comps-lex
- 123 verb-intrans-comp_pp_contra_em-lex
- 124 verb-intrans-comp_pp_afavorde_por-lex
- 125 verb-intrans-comp_pp_afavorde_contra_sobre-lex
- 126 verb-inherent_clitic-comp_pp_sobre-lex
- 127 verb-inherent_clitic-comp_pp_em_por-lex
- 128 verb-inherent_clitic-comp_pp_afavorde_contra-lex
- 129 verb-anticausative-inherent_clitic-comp_pp_para-lex
- 130 verb-anticausative-inherent_clitic-comp_pp_de-lex

Below we list 45 additional types that are known to the grammar but that do not occur in the DeepBank corpus.

- verb-comp_cp+conj_declarative-lex
- verb-comp_cp_declarative-lex
- verb-comp_cp+ind_declarative-comp_indir-lex
- verb-comp_cp_interrogative-lex
- verb-dir_trans-opaque-lex
- verb-inherent_clitic-comp_pp_a_contra-lex
- verb-inherent_clitic-comp_pp_com_de-lex
- verb-inherent_clitic-comp_pp_contra_de-lex

verb-inherent_clitic-comp_pp_perante-lex
verb-intrans-comp_adv-lex
verb-intrans-comp_pp_a_em_sobre-lex
verb-intrans-comp_pp_afavorde_para-lex
verb-intrans-comp_pp_a_para-lex
verb-intrans-comp_pp_àvoltade-lex
verb-intrans-comp_pp_com_contra_em-lex
verb-intrans-comp_pp_com_em-lex
verb-intrans-comp_pp_com_para-lex
verb-intrans-comp_pp_com_sobre-lex
verb-intrans-comp_pp_contra_de-lex
verb-intrans-comp_pp_contra_de_sobre-lex
verb-intrans-comp_pp_contra-lex
verb-intrans-comp_pp_contra_sobre-lex
verb-intrans-comp_pp_de_para_sobre-lex
verb-intrans-comp_pp_em_entre_para-lex
verb-intrans-comp_pp_em_por-lex
verb-intrans-comp_pp_entre_sobre-lex
verb-intrans-comp_pp_perante-lex
verb-trans-comp_pp_a_com_contra_em-lex
verb-trans-comp_pp_a_com_de-lex
verb-trans-comp_pp_a_com_em-lex
verb-trans-comp_pp_a_contra_em-lex
verb-trans-comp_pp_a_contra-lex
verb-trans-comp_pp_a_de-lex
verb-trans-comp_pp_a_para-lex
verb-trans-comp_pp_a_por-lex
verb-trans-comp_pp_com_em-lex
verb-trans-comp_pp_contra_de-lex
verb-trans-comp_pp_contra-lex
verb-trans-comp_pp_contra_para-lex
verb-trans-comp_pp_de_em_por-lex
verb-trans-comp_pp_de_em_sobre-lex
verb-trans-comp_pp_de_para-lex
verb-trans-comp_pp_em_sobre-lex
verb-trans-comp_pp_para_por-lex
verb-trans-comp_pp_perante-lex

Appendix B

Verb lexicon

To complement the description given in §3.1 (page 35), below we give a more thorough account of the verb lexicon used by LX-Gram.

As usual, the asterisk is used to mark the ungrammatical examples.

- Lemma. This field holds the lemma of the word.
- Foreign word? This field holds a yes/no value that indicates whether the word is a foreign word.
- Associated deverbal nouns. This field holds a list of nouns that are derived from the verb form. For instance, *absorver* (Eng.: to absorb) is associated with *absorção* (Eng.: absorption).
- Associated deverbal adjectives. This field holds a list of adjectives that are derived from the verb form. For instance, *absorver* (Eng.: to absorb) is associated with *absorvente* (Eng.: absorptive).
- Variety. This field has three possible values: PO, PE or PB. LX-Gram is able to account for differences between the European and Brazilian variants of Portuguese. As such, entries in the lexicon must be categorized as whether they belong only to European Portuguese (PE), only to Brazilian Portuguese (PB) or to either (PO) variant.
- Is the verb passivizable? This field holds a yes/no value that indicates whether the verb can form passive constructions. For instance, the verb *to see* is passivizable while *to arrive* is not.

- (1) a. Passivizable: *ver* (Eng.: to see)
 (i) O gato viu o pássaro
 the cat saw the bird
 (ii) The pássaro foi visto pelo gato
 the bird was seen by-the cat
 b. Not passivizable: *chegar* (Eng.: to arrive)
 (i) O correio chegou
 the mail arrived
 (ii) *O correio foi chegado
 the mail was arrived
- Alternation. This field describes how causative-anticausative alternations are formed. It has 14 possible values. For the sake of simplicity we show only a few. For instance, “ARG2 to ARG1 oblig. -se” means that when forming an alternation, ARG2 becomes ARG1 and there is an obligatory “se” clitic.
- (2) a. ARG2 to ARG1 oblig. -se: *chatear* (Eng.: to annoy)
 (i) [ARG1 A discussão] chateou [ARG2 a Maria]
 [ARG1 the argument] annoyed [ARG2 Maria]
 (ii) [ARG1 A Maria] chateou-se
 [ARG1 Maria] annoyed-SE
 (iii) *[ARG1 A Maria] chateou
 [ARG1 Maria] annoyed
 b. ARG2 to ARG1 optional -se: *afundar* (Eng.: to sink)
 (i) [ARG1 O pirata] afundou [ARG2 o barco]
 [ARG1 the pirate] sank [ARG2 the boat]
 (ii) [ARG1 O barco] afundou-se
 [ARG1 the boat] sank-SE
 (iii) [ARG1 O barco] afundou-se
 [ARG1 the boat] sank
 c. ARG1 + ARG2 oblig. -se: *divorciar* (Eng.: to divorce)
 (i) [ARG1 O Pedro] divorciou-se [ARG2 da Maria]
 [ARG1 Pedro] divorced-SE [ARG2 Maria]
 (ii) [ARG1 O Pedro e a Maria] divorciaram-se
 [ARG1 Pedro and Maria] divorced-SE
 (iii) *[ARG1 O Pedro e a Maria] divorciaram
 [ARG1 Pedro and Maria] divorced

-
- Acceptable in absolute participles? This field holds a yes/no value that indicates whether the verb can form absolute participle constructions.

- (3) a. Yes: *chegar* (Eng.: to arrive)
- (i) O João chegou e a Paula desmaiou
João arrived and Paula fainted
- (ii) Chegado o João, a Paula desmaiou
arrived João, Paula fainted
- b. No: *sorrir* (Eng.: to smile)
- (i) O João sorriu e a Paula desmaiou
João smiled and Paula fainted
- (ii) *Sorrindo o João, a Paula desmaiou
smiled João, Paula fainted

- Does the verb have a “se” inherent clitic? This field has three possible values: obligatory, optional or not allowed.

- (4) a. Obligatory: *suicidar* (Eng.: to suicide)
- (i) *Ele suicidou
he suicided
- (ii) Ele suicidou-se
he suicided-SE
“He committed suicide”
- b. Optional: *derreter* (Eng.: to melt)
- (i) A manteiga derreteu
the butter melted
- (ii) The butter derreteu-se
the butter melted-SE
- c. Not allowed: *sorrir* (Eng.: to smile)
- (i) Ele sorriu
he smiled
- (ii) *Ele sorriu-se
he smiled-SE

- Subject. This field describes the form of the subject. It has 14 possible values. Again, for the sake of simplicity, we show only a few.

- (5) a. Noun Phrase: *ver* (Eng.: to see)
- (i) [NP O gato] viu o rato
[NP the cat] saw the mouse
- b. Expletive: *chover* (Eng.: to rain)

- (i) Choveu
rained
“It rained”
- c. Raised: *poder* (Eng.: to can)
 - (i) O gato pode ver o rato
the cat can see the mouse
 - (ii) Pode chover
can rain

- Raising. This field holds a yes/no value that indicates whether the verb is a raising verb. This property is related to the semantic interpretation of the verb. With a raising verb, like *começar* (Eng.: to begin), the statement “the surgeon began operating on the patient” implies “the patient began being operated on by the surgeon”; while with a non-raising verb, like *quer* (Eng.: to want), the statement “the surgeon wants to operate on the patient” does not imply “the patient wants to be operated on by the surgeon”.

- Control Reference, Controller and Controlee are three separate fields that describe the behaviour of control verbs. These fields have, respectively, 5, 8 and 3 possible values. As before, we show only a few of them.

- (6) a. Control Reference: Obligatory:
 - querer* (Eng.: to want)
 - (i) O Pedro₁ quer (PRO₁) fazer isso
Pedro₁ wants to (PRO₁) do that
 - (ii) O Pedro₁ quer (PRO₂) fazerem isso
Pedro₁ wants to (PRO₂) do_{plural} that
- b. Controller: Indirect Object:
 - deixar* (Eng.: to let)
 - (i) Os pais₂ deixaram o Rui₁ (PRO₁) viajar
the parents₂ allowed Rui₁ (PRO₁) travel
 - (ii) *Os pais₂ deixaram o Rui₁ (PRO₂) viajarem
the parents₂ allowed Rui₁ (PRO₂) travel_{plural}

- Referential opacity. This property is related with the semantic interpretation of the verb. It can have two values, transparent and opaque. The difference will be explained through the following scenario: Say that John mistakenly believes that Antonio Salieri is the composer of Requiem (the actual composer is Mozart). With a transparent verb,

like *to see*, the statement “John saw Mozart” implies “John saw the composer of Requiem”. However, with an opaque verb, like *believe*, the implication is not valid. For instance, “John believes that Mozart arrived” does not imply “John believes that the composer of Requiem arrived” since John mistakenly believes that Salieri is the composer.

- Clitic climbing. A yes/no value that indicates whether the verb attracts the argument of the embedded verb. In the examples below, the clitic “te” is the argument of *lavar*.

- (7) a. Yes: *estar* (Eng.: to be)
- (i) Estou a lavar-te
i-am washing-TE
“I’m washing you”
- (ii) Estou-te a lavar
i-am-TE washing
- b. No: *acabar* (Eng.: to finish)
- (i) Acabei de lavar-te
i-finished washing-TE
“I’ve finished washing you”
- (ii) *Acabei-te de lavar
i-finished-TE washing

- Complement 1, 2 and 3. These are three separate fields that, together with the Subject field described above, essentially describe the subcategorization frame of the verb. Each of these three fields can have 419 different values. As before, we show only a few examples. For instance, “NP, com NP/de NP” means that the first complement is a NP, the second is a PP (*com* or *de*, followed by a NP), and that there is no third complement.

- (8) a. NP, com NP/de NP:
abastecer (Eng.: to fuel)
- (i) Ele abasteceu [NP o carro] [PP com gasolina]
he fueled [NP the car] [PP with gasoline]
- b. com NP, contra NP:
conspirar (Eng.: to conspire)
- (i) Ele conspirou [PP com eles] [PP contra ela]
he conspired [PP with them] [PP against her]
- c. NP, em NP-Loc/AdvP-Loc:
depositar (Eng.: to deposit)

B. VERB LEXICON

- (i) Ele depositou [NP o dinheiro] [PP no banco]
he deposited [NP the money] [PP in-the bank]
- d. NP, a NP-Dat, para NP:
recomendar (Eng.: to recommend)
 - (i) Ele recomendou [NP a estratégia] [NP a mim]
he recommended [NP the strategy] [NP to me]
[PP para o jogo]
[PP for the game]

We draw attention to (8-c) and (8-d) where the complements are constrained also in terms of their semantic role: Location and Dative, respectively.

References

- AGUSTINI, ALEXANDRE, 2006. *Aquisição Automática de Subcategorização Sintático-Semântica e sua Utilização em Sistemas de Processamento de Língua Natural*. Ph.D. thesis, Universidade Nova de Lisboa.
- BALDWIN, TIMOTHY, 2005. Bootstrapping Deep Lexical Resources: Resources for Courses. In TIMOTHY BALDWIN, ANNA KORHONEN AND ALINE VILLAVICENCIO, editors, *Proceedings of the ACL-SIGLEX Workshop on Deep Lexical Acquisition*, pages 67–76.
- BALLESTEROS, MIGUEL AND JOAKIM NIVRE, 2012. MaltOptimizer: A System for MaltParser Optimization. In *Proceedings of the 8th Language Resources and Evaluation Conference (LREC)*.
- BANGALORE, SRINIVAS AND ARAVIND JOSHI, 1994. Disambiguation of Super Parts of Speech (or Supertags): Almost Parsing. In *Proceedings of the 15th Conference on Computational Linguistics (COLING)*, pages 154–160.
- BANGALORE, SRINIVAS AND ARAVIND JOSHI, 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2):237–265.
- BANKO, MICHELE AND ERIC BRILL, 2001a. Mitigating the Paucity of Data Problem: Exploring the Effect of Training Corpus Size on Classifier Performance for NLP. In *Proceedings of the 1st Human Language Technology (HLT) Conference*.
- BANKO, MICHELE AND ERIC BRILL, 2001b. Scaling to Very Very Large Corpora for Natural Language Disambiguation. In *Proceedings of the*

REFERENCES

- 39th Annual Meeting of the Association for Computational Linguistics (ACL) and 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 26–33.
- BARRETO, FLORBELA, ANTÓNIO BRANCO, EDUARDO FERREIRA, AMÁLIA MENDES, MARIA FERNANDA NASCIMENTO, FILIPE NUNES AND JOÃO SILVA, 2006. Open Resources and Tools for the Shallow Processing of Portuguese: The TagShare Project. In *Proceedings of the 5th Language Resources and Evaluation Conference (LREC)*, pages 1438–1443.
- BARWISE, JON AND ROBIN COOPER, 1981. Generalized Quantifiers and Natural Language. *Linguistics and Philosophy*, 4:159–219.
- BENDER, EMILY, DAN FLICKINGER AND STEPHAN OEPEN, 2002. The Grammar Matrix: An Open-Source Starter-Kit for the Development of Cross-Linguistically Consistent Broad-Coverage Precision Grammars. In *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th Conference on Computational Linguistics (COLING)*, pages 8–14.
- BENNETT, KRISTIN AND COLIN CAMPBELL, 2000. Support Vector Machines: Hype or Hallelujah? *SIGKDD Explorations*, 2(2):1–13.
- BIKEL, DANIEL, 2002. Design of a Multi-lingual, Parallel-processing Statistical Parsing Engine. In *Proceedings of the 2nd Human Language Technology (HLT) Conference*.
- BLACK, EZRA, STEVE ABNEY, DAN FLICKINGER, CLAUDIA GDANIEC, RALPH GRISHMAN, PHIL HARRISON, DON HINDLE, MITCH MARCUS AND BEATRICE SANTORINI, 1991. A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars. In *Proceedings of the Workshop on the Evaluation of Parsing Systems*, pages 306–311.
- BLUNSOM, PHILIP, 2007. *Structured Classification for Multilingual Natural Language Processing*. Ph.D. thesis, University of Melbourne.
- BRANCO, ANTÓNIO, SÉRGIO CASTRO, JOÃO SILVA AND FRANCISCO COSTA, 2011a. CINTIL DepBank Handbook: Design Options for the Representation of Grammatical Dependencies. Technical Report DI-FCUL-TR-2011-03, University of Lisbon.
- BRANCO, ANTÓNIO AND FRANCISCO COSTA, 2008. A Computational Grammar for Deep Linguistic Processing of Portuguese: LX-Gram, version A.4.1. Technical Report DI-FCUL-TR-08-17, University of Lisbon.

- BRANCO, ANTÓNIO AND FRANCISCO COSTA, 2010. A Deep Linguistic Processing Grammar for Portuguese. In *Proceedings of the 9th Encontro para o Processamento Computacional da Língua Portuguesa Escrita e Falada (PROPOR)*, number 6001 in Lecture Notes on Artificial Intelligence (LNAI), pages 86–89. Springer.
- BRANCO, ANTÓNIO, FRANCISCO COSTA, JOÃO SILVA, SARA SILVEIRA, SÉRGIO CASTRO, MARIANA AVELÃS, CLARA PINTO AND JOÃO GRAÇA, 2010. Developing a Deep Linguistic Databank Supporting a Collection of Treebanks: the CINTIL DeepGramBank. In *Proceedings of the 7th Language Resources and Evaluation Conference (LREC)*, pages 1810–1815.
- BRANCO, ANTÓNIO, JOÃO SILVA, FRANCISCO COSTA AND SÉRGIO CASTRO, 2011b. CINTIL TreeBank Handbook: Design Options for the Representation of Syntactic Constituency. Technical Report DI-FCUL-TR-2011-02, University of Lisbon.
- BRANCO, ANTÓNIO, SARA SILVEIRA, SÉRGIO CASTRO, MARIANA AVELÃS, CLARA PINTO AND FRANCISCO COSTA, 2009. Dynamic Propbanking with Deep Linguistic Grammars. In *Proceedings of the 8th Workshop on Treebanks and Linguistic Theories (TLT)*, pages 39–50.
- BRANTS, THORSTEN, 2000. TnT — A Statistical Part-of-Speech Tagger. In *Proceedings of the 6th Applied Natural Language Processing Conference (ANLP) and the 1st North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 224–231.
- BRANTS, THORSTEN AND OLIVER PLAETH, 2000. Interactive Corpus Annotation. In *Proceedings of the 2nd Language Resources and Evaluation Conference (LREC)*.
- BRENT, MICHAEL, 1991. Automatic Acquisition of Subcategorization Frames from Untagged Text. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 209–214.
- BRENT, MICHAEL, 1993. From Grammar to Lexicon: Unsupervised Learning of Lexical Syntax. *Computational Linguistics*, 19(2):243–262.
- BRESNAN, JOAN, editor, 1982. *The Mental Representation of Grammatical Relations*. MIT Press.
- BRISCOE, TED AND JOHN CARROLL, 1997. Automatic Extraction of Subcategorization from Corpora. In *Proceedings of the 5th Applied Natural Language Processing Conference (ANLP)*, pages 356–363.

REFERENCES

- BUCHHOLZ, SABINE, 1998. Distinguishing Complements from Adjuncts Using Memory-Based Learning. In *Proceedings of the Workshop on Automated Acquisition of Syntax and Parsing at the 10th European Summer School in Logic, Language and Information (ESSLI)*, pages 41–48.
- BUCHHOLZ, SABINE, 2002. *Memory-Based Grammatical Relation Finding*. Ph.D. thesis, Tilburg University.
- CALLMEIER, ULRICH, 2000. PET — A Platform for Experimentation with Efficient HPSG Processing Techniques. *Natural Language Engineering*, 6(1):99–108.
- CARROLL, JOHN, 2004. *The Oxford Handbook of Computational Linguistics*, chapter 12, Parsing. In Mitkov (2004).
- CARTER, DAVID, 1997. The TreeBanker: A Tool for Supervised Training of Parsed Corpora. In *Proceedings of the ACL Workshop on Computational Environments for Grammar Development and Grammar Engineering*.
- CASTRO, SÉRGIO, 2011. *Developing Reliability Metrics and Validation Tools for Datasets with Deep Linguistic Information*. Master’s thesis, University of Lisbon.
- CHAWLA, NITESH, KEVIN BOWYER, LAWRENCE HALL AND W. PHILIP KEGELMEYER, 2002. SMOTE: Synthetic Minority Over-sampling TEchnique. *Journal of Artificial Intelligence Research*, 16:341–378.
- CLARK, STEPHEN AND JAMES CURRAN, 2003. Log-Linear Models for Wide-Coverage CCG Parsing. In *Proceedings of the 8th Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 97–104.
- CLARK, STEPHEN AND JAMES CURRAN, 2004. The Importance of Supertagging for Wide-Coverage CCG Parsing. In *Proceedings of the 20th Conference on Computational Linguistics (COLING)*, pages 282–288.
- CLARK, STEPHEN AND JAMES CURRAN, 2007. Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models. *Computational Linguistics*, 33:493–552.
- COLLINS, MICHAEL AND NIGEL DUFFY, 2002. New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 263–270.

- COPESTAKE, ANN, 2002. *Implementing Typed Feature Structure Grammars*. CSLI Publications.
- COPESTAKE, ANN, DAN FLICKINGER, CARL POLLARD AND IVAN SAG, 2005. Minimal Recursion Semantics: An Introduction. *Research on Language and Computation*, 3:281–332.
- CRISTIANINI, NELLO AND JOHN SHAWE-TAYLOR, 2000. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press.
- DAELEMANS, WALTER, JAKUB ZAVREL, KO VAN DER SLOOT AND ANTAL VAN DEN BOSCH, 2009. TiMBL: Tilburg Memory-Based Learner. Technical Report ILK 09-01, Tilburg University.
- DIPPER, STEFANIE, 2000. Grammar-Based Corpus Annotation. In *Proceedings of the Workshop on Linguistically Interpreted Corpora*, pages 56–64.
- DRIDAN, REBECCA, 2009. *Using Lexical Statistics to Improve HPSG Parsing*. Ph.D. thesis, University of Saarland.
- EARLEY, JAY, 1970. An Efficient Context-Free Parsing Algorithm. *Communications of the ACM*, 13(2):94–102.
- EMMS, MARTIN, 2008. Tree-Distance and Some Other Variants of Evalb. In *Proceedings of the 6th Language Resources and Evaluation Conference (LREC)*.
- FAWCETT, TOM, 2006. An Introduction to ROC Analysis. *Pattern Recognition Letters*, 27:861–874.
- FERREIRA, EDUARDO, JOÃO Balsa AND ANTÓNIO BRANCO, 2007. Combining Rule-Based and Statistical Models for Named Entity Recognition of Portuguese. In *Proceedings of the Workshop em Tecnologia da Informação e de Linguagem Natural*, pages 1615–1624.
- FLICKINGER, DAN, 2000. On Building a More Efficient Grammar by Exploiting Types. *Natural Language Engineering*, 6(1):15–28.
- FOUVRY, FREDERIK, 2003. *Robust Processing for Constraint-Based Grammar Formalisms*. Ph.D. thesis, University of Essex.

REFERENCES

- GALAR, MIKEL, ALBERTO FERNANDÉZ, EDURNE BARRENECHEA, HUMBERTO BUSTINCE AND FRANCISCO HERRERA, 2011. An Overview of Ensemble Methods for Binary Classifiers in Multi-Class Problems: Experimental Study in One-vs-One and One-vs-All Schemes. *Pattern Recognition*, 44:1761–1776.
- GAZDAR, GERALD, EWAN KLEIN, GEOFFREY PULLUM AND IVAN SAG, 1985. *Generalized Phrase Structure Grammar*. Harvard University Press.
- GIMÉNEZ, JESÚS AND LLUÍS MÀRQUEZ, 2004. SVMTool: A General POS Tagger Generator Based on Support Vector Machines. In *Proceedings of the 4th Language Resources and Evaluation Conference (LREC)*.
- GIMÉNEZ, JESÚS AND LLUÍS MÀRQUEZ, 2006. *SVMTool: Technical Manual v1.3*. TALP Research Center, LSI Department, Universitat Politècnica de Catalunya.
- HALL, MARK, EIBE FRANK, GEOFFREY HOLMES, BERNHARD PFAHRINGER, PETER REUTEMANN AND IAN WITTEN, 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1):10–18.
- JOACHIMS, THORSTEN, 1998. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *Proceedings of the 10th European Conference on Machine Learning (ECML)*.
- JOACHIMS, THORSTEN, 1999. Making large-Scale SVM Learning Practical. In B. SCHÖLKOPF, C. BURGESS AND A. SMOLA, editors, *Advances in Kernel Methods — Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA.
- JOSHI, ARAVIND, 2004. *The Oxford Handbook of Computational Linguistics*, chapter 26, Tree-Adjoining Grammars. In Mitkov (2004).
- JOSHI, ARAVIND AND YVES SCHABES, 1996. *Handbook of Formal Languages and Automata*, chapter Tree-Adjoining Grammars. Springer-Verlag.
- KAPLAN, RONALD, 2004. *The Oxford Handbook of Computational Linguistics*, chapter 4, Syntax. In Mitkov (2004).
- KAY, MARTIN, 1989. Head-Driven Parsing. In *Proceedings of the 1st International Conference on Parsing Technologies (IWPT)*, pages 52–62.
- KIM, JONG-BOK AND PETER SELLS, 2008. *English Syntax: An Introduction*. CSLI Publications.

- KINGSBURY, PAUL AND MARTHA PALMER, 2003. PropBank: The Next Level of Treebank. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories (TLT)*, pages 105–116.
- KLEIN, DAN AND CHRISTOPHER MANNING, 2003. Fast Exact Inference with a Factored Model for NLP. *Advances in Neural Language Processing Systems*, 15:3–10.
- KORHONEN, ANNA, 2002. *Subcategorization Acquisition*. Ph.D. thesis, University of Cambridge. Published as Technical Report UCAM-CL-TR-530.
- MAGERMAN, DAVID, 1995. Statistical Decision-Tree Models for Parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 276–283.
- MANNING, CHRISTOPHER, 1993. Automatic Acquisition of a Large Subcategorization Dictionary from Corpora. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 235–242.
- MANNING, CHRISTOPHER AND HINRICH SCHÜTZE, 1999. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1st edition. ISBN 0-262-13360-1.
- MARCUS, MITCHELL, MARY MARCINKIEWICZ AND BEATRICE SANTORINI, 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.
- MARQUES, NUNO AND GABRIEL LOPES, 1998. Learning Verbal Transitivity Using LogLinear Models. In *Proceedings of the 10th European Conference on Machine Learning*, number 1398 in Lecture Notes on Artificial Intelligence (LNAI), pages 19–24. Springer-Verlag.
- MARTINS, PEDRO, 2008. *Desambiguação Automática da Flexão Verbal em Contexto*. Master’s thesis, Universidade de Lisboa.
- MATSUZAKI, TAKUYA, YUSUKE MIYAO AND JUN’ICHI TSUJII, 2007. Efficient HPSG Parsing with Supertagging and CFG-Filtering. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1671–1676.
- MCDONALD, RYAN, KOBY CRAMMER AND FERNANDO PEREIRA, 2005. Online Large-Margin Training of Dependency Parsers. In *Proceedings of*

REFERENCES

- the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 91–98.
- MILLER, GEORGE, 1995. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41.
- MITKOV, RUSLAN, editor, 2004. *The Oxford Handbook of Computational Linguistics*. Oxford University Press. ISBN 0-19-927634-X.
- MOSCHITTI, ALESSANDO, 2006. Making Tree Kernels Practical for Natural Language Learning. In *Proceedings of the 11th European Chapter of the Association for Computational Linguistics (EACL)*.
- NIVRE, JOAKIM, JOHAN HALL, SANDRA KÜBLER, RYAN McDONALD, JENS NILSSON, SEBASTIAN RIEDEL AND DENIZ YURET, 2007a. The CoNLL 2007 Shared Task on Dependency Parsing. In *Proceedings of the 11th Conference on Natural Language Learning (CoNLL)*, pages 915–932.
- NIVRE, JOAKIM, JOHAN HALL, JENS NILSSON, ATANAS CHANEV, GÜLSEN ERYIGIT, SANDRA KÜBLER, SVETOSLAV MARINOV AND ERWIN MARSI, 2007b. MaltParser: A Language-Independent System for Data-Driven Dependency Parsing. *Natural Language Engineering*, 13(2):95–135.
- NUNES, FILIPE, 2007. *Verbal Lemmatization and Featurization of Portuguese with Ambiguity Resolution in Context*. Master’s thesis, University of Lisbon.
- OEPEN, STEPHAN, DAN FLICKINGER, KRISTINA TOUTANOVA AND CHRISTOPHER MANNING, 2004. LinGO Redwoods: A Rich and Dynamic Treebank for HPSG. *Research on Language and Computation*, 2:575–596.
- OEPEN, STEPHAN AND DANIEL FLICKINGER, 1998. Towards Systematic Grammar Profiling. Test Suite Technology Ten Years After. *Journal of Computer Speech and Language*, 12(4):411–436.
- OEPEN, STEPHAN, KRISTINA TOUTANOVA, STUART SHIEBER, CHRISTOPHER MANNING, DAN FLICKINGER AND THORSTEN BRANTS, 2002. The LinGO Redwoods Treebank: Motivation and Preliminary Applications. In *Proceedings of the 19th Conference on Computational Linguistics (COLING)*.

- PETROV, SLAV, LEON BARRETT, ROMAIN THIBAU AND DAN KLEIN, 2006. Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 433–440.
- POLLARD, CARL AND IVAN SAG, 1994. *Head-Driven Phrase-Structure Grammar*. The University of Chicago Press.
- PREISS, JUDITA, TED BRISCOE AND ANNA KORHONEN, 2007. A System for Large-Scale Acquisition of Verbal, Nominal and Adjectival Subcategorization Frames from Corpora. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*.
- PRINS, ROBERT AND GERTJAN VAN NOORD, 2003. Reinforcing Parser Preferences Through Tagging. *Traitement Automatique des Langues*, 44:121–139.
- SAG, IVAN AND THOMAS WASOW, 1999. *Syntactic Theory: A Formal Introduction*. CSLI Publications.
- SAMPSON, GEOFFREY AND ANNA BABARCZY, 2003. A Test of the Leaf-Ancestor Metric for Parse Accuracy. *Natural Language Engineering*, 9(4):365–380.
- SAMUELSSON, CHRISTER AND MATS WIRÉN, 2000. *Handbook of Natural Language Processing*, chapter 4, Parsing Techniques. Marcel Dekker. ISBN 0-8247-9000-6.
- SILVA, JOÃO, 2007. *Shallow Processing of Portuguese: From Sentence Chunking to Nominal Lemmatization*. Master’s thesis, University of Lisbon. Published as Technical Report DI-FCUL-TR-07-16 at <http://hdl.handle.net/10455/3095>.
- SILVA, JOÃO AND ANTÓNIO BRANCO, 2012a. Assigning Deep Lexical Types. In *Proceedings of the 15th International Conference on Text, Speech and Dialogue (TSD)*, pages 240–247.
- SILVA, JOÃO AND ANTÓNIO BRANCO, 2012b. Assigning Deep Lexical Types Using Structured Classifier Features for Grammatical Dependencies. In *Proceedings of the Joint Workshop on Statistical Parsing and Semantic Processing of Morphologically Rich Languages at the 50th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 62–71.

REFERENCES

- SILVA, JOÃO, ANTÓNIO BRANCO, SÉRGIO CASTRO AND FRANCISCO COSTA, 2012. Deep, Consistent and also Useful: Extracting Vistas from Deep Corpora for Shallower Tasks. In *Proceedings of the Workshop on Advanced Treebanking at the 8th Language Resources and Evaluation Conference (LREC)*, pages 45–52.
- SILVA, JOÃO, ANTÓNIO BRANCO, SÉRGIO CASTRO AND RUBEN REIS, 2010a. Out-of-the-Box Robust Parsing of Portuguese. In *Proceedings of the 9th Encontro para o Processamento Computacional da Língua Portuguesa Escrita e Falada (PROPOR)*, pages 75–85.
- SILVA, JOÃO, ANTÓNIO BRANCO AND PATRICIA GONÇALVES, 2010b. Top-Performing Robust Constituency Parsing of Portuguese: Freely available in as many ways as you can get it. In *Proceedings of the 7th Language Resources and Evaluation Conference (LREC)*, pages 1960–1963.
- SKUT, WOJCIECH, BRIGITTE KRENN, THORSTEN BRANTS AND HANS USZKOREIT, 1997. An Annotation Scheme for Free Word Order Languages. In *Proceedings of the 5th Applied Natural Language Processing Conference (ANLP)*.
- STEEDMAN, MARK, 2000. *The Syntactic Process*. The MIT Press.
- TJONG KIM SANG, ERIK, 2002. Introduction to the CoNLL 2002 Shared Task: Language-Independent Named Entity Recognition. In DAN ROTH AND ANTAL VAN DEN BOSCH, editors, *Proceedings of the 6th Conference on Natural Language Learning (CoNLL)*, pages 155–158. Morgan Kaufmann Publishers.
- TOUTANOVA, KRISTINA, CHRISTOPHER MANNING, STUART SHIEBER, DAN FLICKINGER AND STEPHAN OEPEN, 2002. Parse Disambiguation for a Rich HPSG Grammar. In *Proceedings of the 1st Workshop on Treebanks and Linguistic Theories (TLT)*, pages 253–263.
- VAN DE CRUYS, TIM, 2006. Automatically Extending the Lexicon for Parsing. In JANNEKE HUITINK AND SOPHIA KATRENKO, editors, *Proceedings of the Student Session of the 18th European Summer School in Logic, Language and Information (ESSLLI)*, pages 180–191.
- VAN NOORD, GERTJAN, 2004. Error Mining for Wide-Coverage Grammar Engineering. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 446–453.

- WING, BENJAMIN AND JASON BALDRIDGE, 2006. Adaptation of Data and Models for Probabilistic Parsing of Portuguese. In *Proceedings of the 7th Encontro para o Processamento Computacional da Língua Portuguesa Escrita e Falada (PROPOR)*, pages 140–149.
- WOLPERT, DAVID, 1992. Stacked Generalization. *Neural Networks*, 5(2):241–259.
- YOUNGER, DANIEL, 1967. Recognition and Parsing of Context-Free Languages in time n^3 . *Information and Control*, 10(2):189–208.
- ZHANG, YI, 2007. *Robust Deep Linguistic Processing*. Ph.D. thesis, University of Saarland.
- ZHANG, YI, STEPHAN OEPEN AND JOHN CARROL, 2007. Efficiency in Unification-Based N -Best Parsing. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT)*, pages 48–59.